

THE ONE ESSENTIAL GUIDE
FOR ANYONE WHO OWNS OR PLANS TO OWN
THE ADAM™ FAMILY COMPUTER SYSTEM

ADAM'S *Companion*

**RAMSEY J. BENSON
and JACK B. ROCHESTER**

PROFESSIONALS
HELP YOU
OVER THE PITFALLS,
GUIDING YOU
STEP-BY-STEP
INTO THE
WONDERFUL
NEW WORLD
OF USING



THE COMPUTER TO ENHANCE YOUR CHILDREN'S EDUCATION...
ORGANIZE YOUR HOUSEHOLD...PLAN YOUR FINANCES...
RUN A SMALL BUSINESS...OR ENJOY THE LATEST GAMES

ADAM'STM ***Companion***

Avon Books are available at special quantity discounts for bulk purchases for sales promotions, premiums, fund raising or educational use. Special books, or book excerpts, can also be created to fit specific needs.

For details write or telephone the office of the Director of Special Markets, Avon Books, Dept. FP, 1790 Broadway, New York, New York 10019, 212-399-1357.

ADAM'STM *Companion*

**RAMSEY J. BENSON
and JACK B. ROCHESTER**



AVON

PUBLISHERS OF BARD, CAMELOT, DISCUS AND FLARE BOOKS

ADAM'S COMPANION is an original publication of Avon Books. This work has never before appeared in book form.

AVON BOOKS

A division of
The Hearst Corporation
1790 Broadway
New York, New York 10019

Copyright © 1984 by Ramsey J. Benson &
Jack B. Rochester
Published by arrangement with the authors
Library of Congress Catalog Card Number: 84-45136
ISBN: 0-380-87650-7

All rights reserved, which includes the right to reproduce this book or portions thereof in any form whatsoever except as provided by the U.S. Copyright Law. For information address Avon Books.

First Avon Printing, June, 1984

AVON TRADEMARK REG. U.S. PAT. OFF. AND IN
OTHER COUNTRIES, MARCA REGISTRADA, HECHO EN
U.S.A.

Printed in the U.S.A.

DON 10 9 8 7 6 5 4 3 2 1

The authors and publishers of this book have used their best efforts in preparing this book and the programs contained in it. These efforts include the development, research and testing of these programs to determine their effectiveness. The authors and publishers make no warranty of any kind, expressed or implied, with regard to these programs, the text, or the documentation contained in this book. The authors and publishers shall not be liable in any event for claims of incidental or consequential damages in connection with or arising out of the furnishing, performance, or use of the text or the programs. The programs contained in this book are intended for the use of the original purchaser-user.

COPYRIGHTS AND TRADEMARKS

ADAM™, ADAMLink™, ColecoVision®, SmartBASIC™, SmartFiler™, Smart-Modem™, SmartPICTURE PROCESSOR™, SmartWriter™, and Super Action™ are trademarks of Coleco Industries, Inc. © 1983, 1984 Coleco Industries, Inc.

APPLE™ and APPLESOFT™ are trademarks of Apple Computer, Inc.

CP/M® is a trademark of Digital Research Corporation.

GORF™ is a trademark of Bally Midway Mfg. Co. © 1980 Bally Midway Mfg. Co. All rights reserved.

OMEGA RACE™ is a trademark of Bally Midway Mfg. Co. © 1981 Bally Midway Mfg. Co. All rights reserved.

TELLY TURTLE™ and BRAIN STRAINERS™ are trademarks of Carousel Software, Inc. © 1983 Carousel Software, Inc.

SLITHER™ © Century II.

Exidy's MOUSE TRAP™, PEPPER II™ and TIME PILOT™ are the trademarks of Exidy Incorporated © 1981 Exidy Incorporated.

VENTURE™ and VICTORY™ are the trademarks of Exidy Incorporated © 1982 Exidy Incorporated.

DONKEY KONG and DONKEY KONG JUNIOR are trademarks of Nintendo of America, Inc. © 1982 Nintendo of America, Inc.

SMURF™ and GARGAMEL™ are the trademarks of Peyo. © 1983 Peyo Licensed by Wallace Berrie & Co. Van Nuys, CA.

PAINT' N' PLAY THEATER™ is a trademark of Peyo. © 1984 Peyo Licensed by Wallace Berrie & Co. Van Nuys, CA.

BUCK ROGERS™ indicates a trademark of The Dille Family Trust.

CARNIVAL® and SEGA® are the trademarks of SEGA Enterprises, Inc. © 1980 SEGA Enterprises, Inc.

SPACE FURY™, TURBO™ and SEGA® are the trademarks of SEGA Enterprises, Inc. © 1982 SEGA Enterprises, Inc.

PLANET OF ZOOM™, SUBROC™, ZAXXON™ and SEGA® are the trademarks of SEGA Enterprises, Inc. © 1982 SEGA Enterprises, Inc.

SPACE PANIC™ is a trademark of Universal Co., Ltd. © 1980 Universal Co., Ltd.

COSMIC AVENGER™ and LADY BUG™ are the trademarks of Universal Co., Ltd. © 1981 Universal Co., Ltd.

MR. DO!™ is a trademark of Universal Co., Ltd. © 1982 Universal Co., Ltd.

LOOPING™ is a trademark of Venture Line, Inc. © 1982 Venture Line, Inc.

STERN® and FRENZY™ are the trademarks of Stern Electronics, Inc. © 1982 Stern Electronics, Inc.

WILD WESTERN™ is a trademark of Taito America Corporation © 1982 Taito America Corporation.

TARZAN™ Owned by EDGAR RICE BURROUGHS, INC. and Used by Permission. Copyright © 1984 EDGAR RICE BURROUGHS, INC. and Coleco Industries, Inc. All rights reserved.

ROCKY™ © 1982 United Artists Corporation. All rights reserved.

WAR GAMES™ © 1983 United Artists Corporation. All rights reserved.

THE DUKES OF HAZZARD* * indicates trademark of Warner Bros., Inc. © 1984 Warner Bros., Inc.

DRAGON'S LAIR™ Owned by MAGICOM, INC. and Used by Permission. © 1983 MAGICOM, INC. All rights reserved.

TUNNELS 'N' TROLLS™ is a trademark of Flying Buffalo, Inc. © 1979 Flying Buffalo, Inc. All rights reserved.

HAPPY BIRTHDAY TO YOU by Mildred and Patty Hill © 1935 Summy-Birchard Music division of Birch Tree Group Ltd. Used by Permission.

This book is lovingly dedicated to our sons,

GREGG R. BENSON
and
JOSHUA B. ROCHESTER

The only difference between men and boys
is the price of their toys.

ACKNOWLEDGMENTS

"True ease in writing comes from art, not chance," said the poet Alexander Pope, to which we would add a dollop of hard work. Writing a book is arduous under the best circumstances, but to produce a useful, accurate book about a brand new computer in two and a half months is something else again. First and foremost, we would like to thank our wives, Dolores Benson and Mary Driscoll Rochester, who put up with our moods, absences, long hours and the hectic pace we kept while writing *Adam's Companion*.

We would also like to thank Garrow Throop and Kristine Nappi of Garografiks, Inc., Newton, Massachusetts, for their graphic design work and Jonas Gedraitis of Brockton, Massachusetts, for his artful renderings. Thanks also to Taylor Barcroft, founder of the First Southern California Adam Users Group and publisher of the *Garden of Adam Newsletter*, for his encouragement, counsel and contributions to Appendix E.

When we first discussed the project with our editor, John Douglas, he asked if we thought Coleco would support our efforts. We replied, somewhat philosophically, that it could go one of two ways: either with 'em or without 'em. As it turned out, Coleco went the extra mile for us every time. Extra special thanks to the following people:

Mark L. Yoseloff, executive vice-president
Al Kahn, executive vice-president, marketing
Morton Handel, executive vice-president, finance
Barbara C. Wruck, director of corporate communications
Karen L. Gershman, coordinator, marketing communications
Joe Mallardi, director of customer service
John Reese, marketing manager for ADAM
Kathy McGowan, marketing manager for corporate communications

and the many other Coleco employees for their thoughtful assistance.

And, last but not least, thanks to ADAM, for being a computer worthy of all our efforts.

RAMSEY J. BENSON
Staten Island, New York

JACK B. ROCHESTER
Millis, Massachusetts

January 1984

TABLE OF CONTENTS

CHAPTER 1	Introducing ADAM	1
CHAPTER 2	How to Write and Print a Perfect Letter in Six Easy Steps with SmartWriter	16
CHAPTER 3	Getting the Most from Your ADAM Computer	25
CHAPTER 4	A Buyer's Guide to ColecoVision Games	35
CHAPTER 5	How to Design and Write Your Own BASIC Programs	62
CHAPTER 6	More Things You Can Do with SmartWriter	101
CHAPTER 7	How to Create Your Own Picture Maker	124
CHAPTER 8	Writing Your First Video Game in BASIC	166
CHAPTER 9	How to Write Music on ADAM (Even if You Can't Read a Single Note)	189
CHAPTER 10	How to Write a Mailing-Label System in BASIC	223
CHAPTER 11	ADAM's Future	262
APPENDIX A:	Description of ASCII Character Codes	270
APPENDIX B:	SmartBASIC Command and Function Summary	275
APPENDIX C:	List of Books and Magazines	379
APPENDIX D:	How to Create a Shape Table	383
APPENDIX E:	How to Start Your Own Computer Club	392

ADAM'STM ***Companion***



CHAPTER ONE

Introducing ADAM

This is a book about the ADAM Family Computer System, the most powerful, fully integrated computer ever made for the American family. In the past, personal or home computers were either very expensive or incomplete. ADAM is neither. You can take it out of the box and begin using it to write letters, manage information, learn the BASIC programming language or play games.

Adam's Companion tells you everything you need to know about ADAM, whether you are planning to purchase one or if you already have. Each chapter discusses an aspect of ADAM in great detail to



FIGURE 1-1 ADAM Family Computer System.

help you learn to use your computer correctly and to help you utilize it to its fullest potential.

Everything you read about in this book is a result of the authors' direct, hands-on experience with ADAM. We have tried and tested all the applications and programs to be sure they work properly for you. When there is a problem or something you should watch for, we point it out. We've also shared many of our thoughts from the Common Sense Department, based on our long experience with computers. Although we encourage you to experiment with ADAM and find new uses for your computer, don't get creative with cords, plugs or taking the machine apart. ADAM is a sophisticated electronic instrument and a real computer, not a toy. Although it's designed for rugged use, it requires tender, loving care. You'll find that writing BASIC programs is another area where it's important to follow the rules precisely. If you're careful with your ADAM, it should be a good friend to you and your family for many years.

ADAM comes in two different packages: as a complete Family Computer System and as Expansion Module #3. The latter is designed for use in conjunction with the ColecoVision Video Game System, if you already have one. Maybe they should have called it Eve!

The expansion module provides the printer, keyboard and memo-

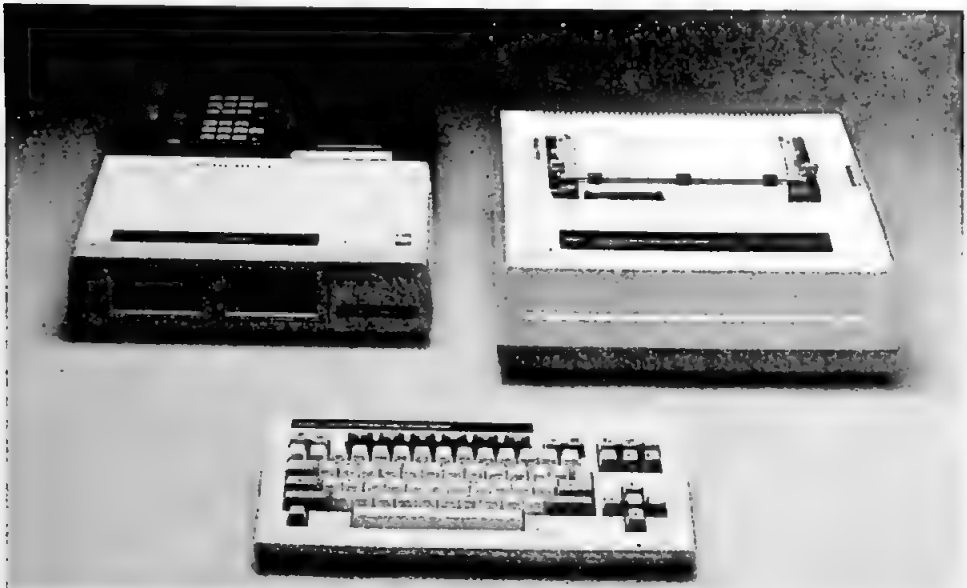


FIGURE 1-2 ColecoVision Expansion Module #3.

ry console, which fits together with the ColecoVision game system, as well as all the cords, parts and manuals. The two units mount on a plastic base that is nearly 24" deep, which means it requires quite a bit of tabletop space. Since your ColecoVision comes with game controllers, they aren't included with Expansion Module #3. You end up with a fully functional ADAM, but in the process you save some money. However, if your family uses the ColecoVision a lot and you think ADAM will get heavy use as well, you might want to keep your game system and buy ADAM complete.

Hardware

Coleco's analogy between Adam, the first man God created, and ADAM the computer is not a bad one. Computers are more like human beings than many would like to admit. The brain, the nervous system and our physical bodies work together to accomplish what we want to do. Likewise, three parts of the computer system work together so it can perform tasks for you.

The brain without eyes, ears or mouth cannot communicate with the world. A computer's brain is called the central processing unit, or CPU, and it is here that all actual computing takes place. The CPU is a powerful yet tiny microchip about the size of a pea. ADAM's CPU is in the memory console, which houses the data cassette drive and the SmartWriter program, which is inside also on a microchip. If you like, you can open the hatch on top of the memory console and see three sockets, or ports. These ports accept printed circuit boards that, in the future, will allow ADAM to do other tasks for you. More on this later.

Yet without a keyboard, video monitor or printer, the CPU is quite useless. The keyboard and printer that come with your ADAM, along with the video monitor or television set, are called *input/output devices* because they put data into the CPU or take it out. The keyboard or game controller allows you to *input*—or change what's happening in the computer (write letters, move Buck Rogers's space ship): you can *output*—or see what's happening on the video screen or print out your letters on the printer. These devices are also called *peripherals*, because they are secondary (although obviously quite essential) to the computer's primary function.

Software

Without intelligence, the brain cannot direct the body into useful activities. Programs, usually called software, give the computer intelligence. Programs are simply instructions written in a language the computer understands, recorded on the data cassettes in much the same way you record music on your stereo. Instead of getting an album like *Men at Work*, you get a program. Programs aren't always on cassettes, though; everything on cassette is also available on floppy disks if you have the disk drive. In addition, SmartWriter is electronically recorded on a microchip, very similar to the CPU chip, so that you automatically get the SmartWriter program every time you switch ADAM on. This means you don't have to wait for the program to load from the cassette, which can be pretty handy. In the future, Coleco will provide certain other programs on chips for you.

Data Cassettes and Floppy Disks

Both programs and the information, or data, you want to save are kept on what we call *magnetic storage media*. Your ADAM comes with one cassette drive; you can add a second if you wish. One data cassette holds about 250 typewritten double-spaced pages. On the other hand, a floppy disk holds one and one-half times as much—360 pages!

Data cassettes and floppy disks perform two jobs for you. One, they hold prerecorded programs such as SmartBASIC, SmartLOGO or Typing Tutor. You put the cassette or disk in the drive, press RESET, and in a few moments you've filled the computer's brain (the CPU, remember?) with intelligence. Two, they provide a way for you to save the letters, term papers, BASIC programs or other important information you write. Once ADAM has loaded the program from one cassette or disk, you can take it out and put a fresh one in its place for recording and storing data. Floppy disks retrieve and store programs and information much faster than data cassettes.

It's important that you take good care of your cassettes or disks, and your game cartridges as well. Here are a few tips:

- Return the cassette to its plastic storage box promptly after use to avoid dirt, dust and cookie crumbs.
- Never touch the tape or disk surface. The natural oil in your skin can damage the surface, causing you to lose data. If a loop develops in

the tape, insert a pencil in the gray plastic spoked hub and rewind it carefully until the tape is snug.

- Be careful never to expose your cassettes or disks to magnetic fields, such as electric motors (even the one in your printer!), your television, electric typewriter, air conditioner, electric fan, stereo speakers or, of course, magnets.

- Beware of static electricity as well. Sometimes people pick up static charges when they walk across the carpeting; if you do, don't do it while holding a cassette, because you'll zap the data right off the tape! If your home seems especially prone to static electricity, buy a can of antistatic spray for your carpeting.

- Don't expose cassettes or disks to temperature extremes. Heat will warp the plastic case, and intense cold will snap the tape or disk.

To insert a cassette, open the drive door by pressing the EJECT button on top away from you. Drop the cassette in with the writing toward you and the side with tape exposed down. Close the door and the tape will be mounted. To remove the cassette, press the EJECT button to open the door. If the cassette does not pop free, reach inside with your index finger and gently pull one side of the cassette toward you. It will pop right out. You can safely remove a cassette any time the tape is not spinning.

The disk drive, mounted on top of the memory console, loads and unloads much like a cassette in your car stereo. Since it's sealed in a plastic case that only goes in one way, you can't do much to harm it, but follow the instructions Coleco provides with your disk drive unit carefully nevertheless.

If you're like most people, you'll probably accumulate many cassettes or disks and a fair number of cartridges for ADAM. You might consider purchasing a storage box for them, like the computing professionals use for their floppy disks. There are quite a few good storage boxes on the market made of high-impact, static-free plastic. Most computer stores and discount department stores carry storage boxes with hinged lids that sell for \$15-\$30. A best buy is the Add'nStac interlocking cassette and game cartridge modules. We bought ours in 3-packs for \$4.99. If you can't find them, contact Royal Sound Company, Inc., 200 Industrial Way, Eatontown, NJ 07724. (201) 542-8400. It's a good idea to clean your tape drive, with normal usage about once a month. Radio Shack sells a kit for just a few dollars (Catalog #44-1170) containing solvent and long cotton swabs that reach far enough to clean the head and tape path. You should

clean the head (the bright chrome device with four black squares on it) *very carefully to avoid scratching it*, and the rubber wheel just to the right. Mop up any dust or tape particles you see lying inside the tape drive. Don't attempt to clean the floppy disk drive; it requires far less attention than a tape drive, and when maintenance is needed it should be done by a trained service technician.

Conducting the Computer Orchestra

The brain understands everything it sees or hears, but it controls the body's movements as well with its nervous system. ADAM coordinates between its programs and the three peripherals with an *operating system*, which is very similar in function to the nervous system. For example, the operating system tells the tape drive to load the program or instructs the printer to print out a letter. You probably won't ever handle the operating system, and because it's a set of instructions (very much like a program) recorded on a chip it's unlikely that it will ever break, but it's still good to know when it does. Without it, your ADAM simply would not work.

A computer is no less amazing than a human—perhaps more so because humans invented the computer. The first practical working computer was the ENIAC, which stands for Electronic Numerical Integrator and Calculator, completed in 1946. It weighed 30 tons, contained 18,000 glowing electron tubes, and occupied a large room. Your ADAM can perform more tasks, and can do them 20 times faster, than old ENIAC. Look at the progress we've made!

This is really about all you need to know to understand how your ADAM works. For those of you who are seriously interested in learning more about the science of computing and the art of programming, the chapters covering BASIC go into a great deal more depth explaining bits, bytes, and stuff like that. But right now, let's make sure you have your computer properly hooked up.

CORDS, WIRES, SWITCHES AND STUFF

ADAM isn't hard to connect, and we recommend you read the *Set-Up Manual* that comes with it. There are a few things, however, that may



FIGURE 1-3 ENIAC, Moore School of Electrical Engineering, University of Pennsylvania, 1946.

help you in the process. When you open the big box ADAM comes in, take the printer out first. Be sure to remove the foam blocks from either side of the print head or you won't do much printing! Make sure the power switch is off and plug the outlet into the wall.

You probably noticed the switch is on the back of the printer. We think it's just as easy to turn it around, rear end toward you, to get at that switch. You must turn ADAM off each time you change cartridges, and if you use continuous-form paper it feeds from the back anyway, so you might consider this solution.

Protect Your ADAM

Another alternative is to buy a power-controller box. This is a wise move, because these boxes have circuits inside that protect against power surges, brownouts and sudden changes in voltage—all of which happen all the time. A power surge could cause serious damage

to ADAM's electronic components, or could cause you to lose valuable data that isn't stored on a data cassette.

There are a number of very good power-controller boxes on the market, and there is very little difference between them. If you plan on adding more components to your ADAM over a period of time, we recommend you buy one with the following features:

- Six outlets
- Six-foot power cord
- Master power switch located on the box itself
- Replaceable fuse or resettable circuit breaker

Prices for these units range from \$70 to \$140, a bit expensive, but you only have to have one power failure to be glad you bought one. You can shop the stores or look in computer magazines for good buys, but for convenience as well as price, we rate Radio Shack's Auto Control Power Strip and Surge Protector (Catalog #26-1429) a best buy. Note that you can purchase a power strip that does *not* have the filtering and surge protection circuits inside. These sell for about \$20-\$30 and offer convenience only, no protection.

If you don't think you're going to need six outlets and simply want to protect ADAM, you should buy a one-socket unit. We rate the Panamax Surge Suppressor SS120/1 a best buy. It provides both surge and filtering protection, the same as the more expensive models, and comes with a replaceable fuse and monitor light. As a special offer, you, the reader, can buy the Panamax SS120/1, which lists for \$59, for \$35, postpaid. Send your check to the distributor, along with a note saying you read about this special offer in *Adam's Companion*, to:

Technology Tool Company
55 Wheeler Street
Cambridge, MA 02138

Another alternative, if you're handy with a soldering iron, is to build your own controller. *InCider* magazine (August 1983, page 108) features a kit using parts from Radio Shack that costs \$25 to build yourself. If you have trouble locating that issue, you can write to the author, George M. Engel, 35 Old Ansonia Road, Seymour, CT 06483.

Other Connections

Once you have the printer properly connected to your household current (black cord), plug the gray cord into the memory console. Note the plug is marked TOP and is shaped so it only goes in one way. Be careful to plug it in correctly, because this cord transfers power and signals throughout the computer system. Note that there is a special plug for the ColecoVision roller-ball module that goes between the plug and the outlet on the memory console. This is because the roller-ball module requires electrical power, so be careful to make good connections when you hook it up.

The memory console is so called because the cassette drive and internal memory storage are located here, but it's also where the CPU, ADAM's heart, resides. You can see that all the system components—the keyboard, the printer, TV and game controllers—all plug into the memory console. *Make certain all the cables and cords are properly plugged in before you turn the power on.* You've already connected the printer, so now connect the keyboard, using the coiled cord with modular telephone plugs on each end, into the front of the memory console. There is another phone plug socket on the left side labeled ADAMNET, which is used for the modem, if and when you get one. The game controllers plug, simply enough, into the right-hand side of the console, with controller 1 closest to you.

Directly above the tape drive is a plastic panel with the ADAM nameplate on it. If you open this door and look inside you'll see three plugs, called *expansion slots*, intended for future accessories on *expansion boards or modules* for your ADAM.

You'll also notice how the tape drive is connected inside. If you plan to add a second drive, all you have to do is unscrew the two screws holding the metal plate in place, put the new drive in and replace those screws, and plug the two plugs into their respective outlets. *Do this very carefully—the pins are easily bent.*

Inserting cassettes, such as the Super Game Packs, is easy. Open the drive door by pushing the release tab away from you and drop the cassette, tape-opening side down with the label reading "ADAM High Speed Digital Data Pack" facing you, into the opening. Then simply push the door shut. If it doesn't close, you've put the cassette in either upside down or wrong-side out.

The Video Connection

One of the nice things about ADAM is that it's a complete computer system—almost. You need a television set, preferably color, to get the most of what ADAM offers. Now we come to the back of the memory console, where you make the video connections. (If you are using Expansion Module #3, the following information does not apply. Use the standard video connections on the back of the ColecoVision console for your television.) First, set the switch to select either channel 3 or 4; make certain your television channel selector is set for the corresponding channel. The next socket connects to your *television* set, and goes to the video interface box provided with your ADAM. Instructions for connecting it to your TV antenna leads are in the *Set-Up Manual*.

Most people will simply hook ADAM up to their TV and have perfectly satisfactory service, but we advise you to take a minute to consider a monitor. A monitor works only with a computer—it's not a television—and has much finer detail, providing a clearer, sharper and, if it's a color set, more colorful picture. Because a monitor provides more detail—"high resolution" we call it—you don't get funny-looking letters and characters that look like they're three or four different colors. And, if you add the appropriate Coleco accessory, you'll be able to see all 80 columns of SmartWriter text instead of just the 40 you see on a TV. That means a sheet of electronic paper in SmartWriter will look just as wide as the sheet of paper the printer prints out.

If you plan to use your color television for SmartWriter and the color mosaics bother you, turn the color off so the screen is black and white. You may find it easier on your eyes.

Better yet is a monitor. A good 13"-monitor will cost about the same or perhaps a little more than a television; a green-screen or black-and-white monitor costs between \$100–\$200. Color monitors begin at about \$350–\$450 for a *composite* set, which combines audio and video into one wire and connects to the MONITOR video jack, and go on up in price for an RGB unit. RGB means, simply, that red, green and blue are separated from each other *and* separated from the audio channel. But ADAM generates only a composite signal, not the separate signals required by an RGB monitor.

Using a monitor also means you can use your ADAM somewhere besides the middle of the coffee table in the living room. There are a number of good monitors on the market made by NEC, Comrex,

Zenith, Amdex and Quadram that you can check out in personal computing magazines or at retailers. *Consumer Reports* published an informative article on monitors, with ratings, in the October 1983 issue, condensed and reprinted in the *Consumer Reports 1984 Buying Guide Issue*.

Whatever monitor you decide to buy, make sure it has a connection for a 300-ohm input. Like a television set, monitors may have a 75-ohm, a 300-ohm, or both. ADAM requires a 300-ohm connection to make a perfect picture. Make sure it has a built-in speaker or you won't be able to hear ADAM!

If you're one of those people who take their sights and sounds seriously, another option is component television. These sets separate the monitor from the tuner, which has jacks for plugging in a computer, videocassette recorder and videodisk player, and often provide sound jacks that plug into your stereo system. The advantage is that the video for television, games and computing is real no-compromise state-of-the-art viewing. Sony, Panasonic, Matsushita and RCA make component television systems, usually in 19" and 25" screens, that start at around \$1500—expensive, but worth it.

That leaves only one port, the big one on the right side of the memory console. This is for Coleco's Atari Expansion Interface #1, so you can play your Atari cartridge games on ADAM. If you get Turbo, the Expansion Module #2, it plugs into the two controller ports. Each of these connections is explained in detail in the appropriate owner's manual.

Simple Troubleshooting

It's hard to believe, but 80% of all problems people have with their computers are because something isn't plugged in. You're probably saying to yourself, "Well, that would never happen to me, because *I'd* never forget to plug in the cord." Well, it's happened to us many times, and it could happen to you, too. So if you turn ADAM on—power up, as computer pros say—and nothing happens, *check the cords and wires*. A loose plug can happen to anyone! Here are a few troubleshooting tips:

- When you power up, the red light on the lower right corner of the keyboard should be on. If it's not, check the power cord connections.
- If the red light comes on but you have no video, check the following:

Is the television or monitor power switch on?

Is the TV/COMPUTER video interface switch set to COMPUTER? Are the television and ADAM set for the same channel?

Is the video cable properly plugged into the correct jacks?

- If everything is correctly connected and you still have nothing on the screen, press the RESET switch to the left of the game cartridge slot. This should bring ADAM's electronic typewriter to the screen.

- If you hear an excessive amount of noise or static from your TV speaker, you might try moving the printer a distance away from it; the motor may be causing interference.

If neither these tips nor those in your *ADAM Set-Up Manual* solve your problem, you can call Coleco for help at (800) 842-1225.

A HOME FOR ADAM

Regardless of whether you use a monitor or a television, it's a good idea to give some thought to where you set up your ADAM. Hint: the living room coffee table isn't a great location. Someone is bound to spill a soft drink or a bowl of cereal on it, or accidentally knock it on the floor. ADAM is rugged, but is nonetheless filled with sensitive electronic and mechanical parts. You don't want your computer in the repair shop for three months due to a moment's carelessness, do you? You should also protect ADAM from static electricity, excessive humidity and temperature extremes. Dust is another of the computer's enemies; you might want to cover the keyboard and printer when not in use. Radio Shack Computer Centers sell plastic covers for computer equipment, or you could make attractive cloth covers to match your decor.

A desk or table just for ADAM, of course, is the ideal solution. Your decision will depend, in part, on how much time you spend playing games versus how much you use ADAM for practical work and writing. There are many companies making computer furniture now, such as O'Sullivan, which offers several designs and price ranges beginning at \$100. Cargo Furniture specializes in solid pine, CBO Company in solid oak; J.K. Products is Danish modern, while Presentation Systems is high-tech tubular steel and bright colors. Quality of Life Environments handcrafts custom Shaker-style furniture. So you see, there's something for any taste. Browse through any

of the personal computing magazines or local furniture and computer stores to see what you like.

You might want to jot down ADAM's measurements before you go shopping, just to be sure all the equipment fits. The memory console is a little over 4" high, 11" deep and 18" wide. You'll need plenty of extra room on the sides for plugging and unplugging game controllers and the Atari expansion module. The keyboard is 2" high, 7" deep and 15" wide. Since it usually sits in front of the memory console and can be moved around, you'll want a workspace at least 24"–30" deep for these two components.

The printer is 6" high, 13" deep and 15" wide. You'll need room on top to feed paper in, and if you plan on using continuous-form paper, you'll need another 15" in back of the printer to stack the paper. Often people purchase a box containing 2400 or 3600 sheets of paper, open it and let it sit on the floor beneath the table the printer is nesting on, so that it feeds up through the platen. Coleco plans to sell a *tractor feed*, a device with toothed wheels that clips on the printer and pulls continuous-form paper through. On the present printer, however, you can print out between six and eight sheets of continuous-form paper without it pulling or sliding. Besides, if it looks like the paper is beginning to pull crookedly, simply press STOP PRINT, readjust it, and start up again.

You've probably noticed that when you turn ADAM on, the printer head—the device that strikes the daisy wheel to actually print the letters on the paper—is at the far left side of the platen. When you insert a sheet of paper it will often jam up in the print head; to avoid this, gently slide the assembly to the middle of the platen. It moves easily, and returns to the left margin when you begin the print process, so don't worry about harming it.

You can select from a variety of different daisy-wheel type faces, such as italic, gothic, roman and others. The daisy wheel is easy to change, but be careful of the small, bright-chrome metal tab with the black tip on the left side of the print head; it is easily bent.

The ribbon is a carbon ribbon, which means it can be used only once, unlike the cloth ribbons often used in typewriters, but it makes a sharper, darker impression. Your ribbon cartridge is good for about 500,000 impressions, which is about 200–250 double-spaced pages. The printer automatically stops when the ribbon runs out, to let you know it needs a new one. Be sure to keep at least two extra ribbons around, just in case one jams or is defective (which does happen).

You may find the printer somewhat loud. If the noise level is objectionable, consider the following:

1. Buy a thick acoustic pad, which costs between \$10-\$15, at an office-supply store.

2. Carefully tape or glue pieces of 1" foam inside the printer cabinet (in front of the platen and print head, not in the enclosed area behind).

3. Build an acoustic cover (you can find plans for one in the November 1983 issue of *Family Computing*, "How to Build a Printer Muffler for Under \$20," by Gene and Katie Hamilton).

Care and Feeding of ADAM

Along with making a nice workspace for ADAM, you'll want to make sure your computer is safe and protected. Your homeowner's insurance policy should cover ADAM in case of theft, but it's a good idea to call your insurance agent and make sure you have the proper coverage. Thieves go after televisions and stereos, so we can expect home computers will be topping their hit parade next. If possible, try to keep ADAM where it's not readily visible to thieves from the outside.

Computer Supplies

Coleco has made arrangements for ordering printer ribbons, daisy wheels, cassettes, floppy disks and ADAM accessories using your credit card from their toll-free line, (800) 842-1225. You may also call this number to ask if Coleco has revised or updated the manuals that came with your ADAM; if so, they'll send you free replacement copies. However, if you want stationery, continuous-form paper, self-adhesive labels, preprinted forms, invoices, envelopes and other office supplies, there are several mail-order firms you'll want to contact. All of them will send you a free catalog, and you can order over their toll-free lines. These firms usually ship immediately, and you'll usually receive your order within a few days. Here's a selection:

American Computer Supply, (800) 527-0832

Global Computer Supplies, (800) 645-6393

Misco Computer Supplies & Accessories, (800) 631-2227

Moore Computer Forms & Supplies, (800) 323-6230
New England Business Service, (800) 225-9550
The Reliable Corporation, (800) 621-4344

As an alternative, you may want to order *The Directory of Discount Computer Suppliers*, which lists all mail-order companies that sell discounted computer supplies. It's \$3.50 from Discount America Publications, 51 East 42 Street, Room 417P, New York, NY 10017.

Now that you've learned about hardware and software, how to hook up your ADAM correctly, and how to obtain supplies, let's get started with SmartWriter and see what ADAM can do for you!

8

518-843-4390

1/154 -

CHAPTER TWO

How to Write and Print a Perfect Letter in Six Easy Steps with SmartWriter

"Oh, that my words were now written!," cried Job, "Oh, that they were printed in a book!" In Job's time, a book was a sheaf of papyrus leaves on which one wrote with a quill and ink. Job's words were not actually printed until Johann Gutenberg invented movable type in his print shop in Mainz, Germany, and published the *Mazarin Bible* in 1436.

In the 20th century, most of us have suffered Job's lament, even using typewriters. But word processing has changed everything, offering us the power to transform laborious handwriting and problematic typing into sleek, colorful electronic blips and friendly beeps on our television set.

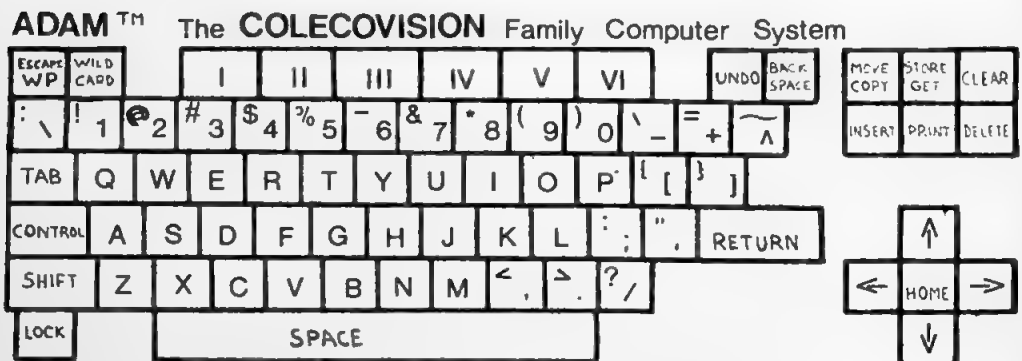


FIGURE 2-1 ADAM's Keyboard.

In a few minutes, you're going to be a word-processing pro. SmartWriter is very simple to learn, and is just as efficient as the big word-processing systems in offices. You'll be able to write crisp business letters and print out professional-looking resumes. If you're a writer, you'll find SmartWriter helps overcome writer's block, because you won't be staring at an intimidating sheet of blank paper any more. Writing letters with SmartWriter is easy, quick and fun, because your words tumble out almost as fast as you think of them! Children will profit by using SmartWriter because it will help them spell better, learn to think and express themselves in the language and improve their writing skills.

SMARTWRITER BY THE NUMBERS

One. Flip the power switch on the back of the printer on. You'll hear a beep and the print head will move to the left end of the platen and then make a striking noise. It doesn't actually print a character on the paper: it's just checking itself to make sure everything works properly. The red power light on the bottom right corner of the keyboard should be on.

Two. On the screen you'll see a picture of a blank sheet of paper and a typewriter-like platen at the bottom. ADAM's screen was designed to look and work just like a typewriter so that, as you type, the words appear on the paper and move up to fill the screen. Below the platen you'll see text that reads ADAM'S ELECTRONIC TYPEWRITER on the left and shows two boxes for setting margins and tabs on the right. ADAM will act just like a typewriter now—every key you press will type that letter or number on the printer and on the screen at the same time. Roll a fresh sheet of paper into the printer and experiment with the electronic typewriter for a minute or two. Try typing this sentence:

I know not what course others may take, but as for me, give

now press the RETURN key

me liberty or give me death!

and press the RETURN key again

You can type as fast as you want and ADAM will keep up with you. Even if you don't see the letters appearing on the screen as you type them, don't worry! ADAM will catch up and every one will print out, even the mistakes you make. Try typing Patrick Henry's famous statement urging the start of the Revolutionary War several times until you get the hang of it.

Three. Press the key in the top left-hand corner of the keyboard marked ESCAPE/WP. You'll hear a friendly chirrup indicating you've stopped using the electronic typewriter and are now in word processing. You'll see the bottom portion of the screen, which we call the *message window*, below the platen, change. There are six functions, one in each square, which correspond to the six function keys (called *smart keys*) at the top of your keyboard. When you want to give SmartWriter instructions or commands, these are the keys you use. The smart keys work in conjunction with the six keys to the right to perform all the word-processing functions. The top portion of the screen where you were typing, the *document window*, still looks the same as it did when you were using the electronic typewriter.

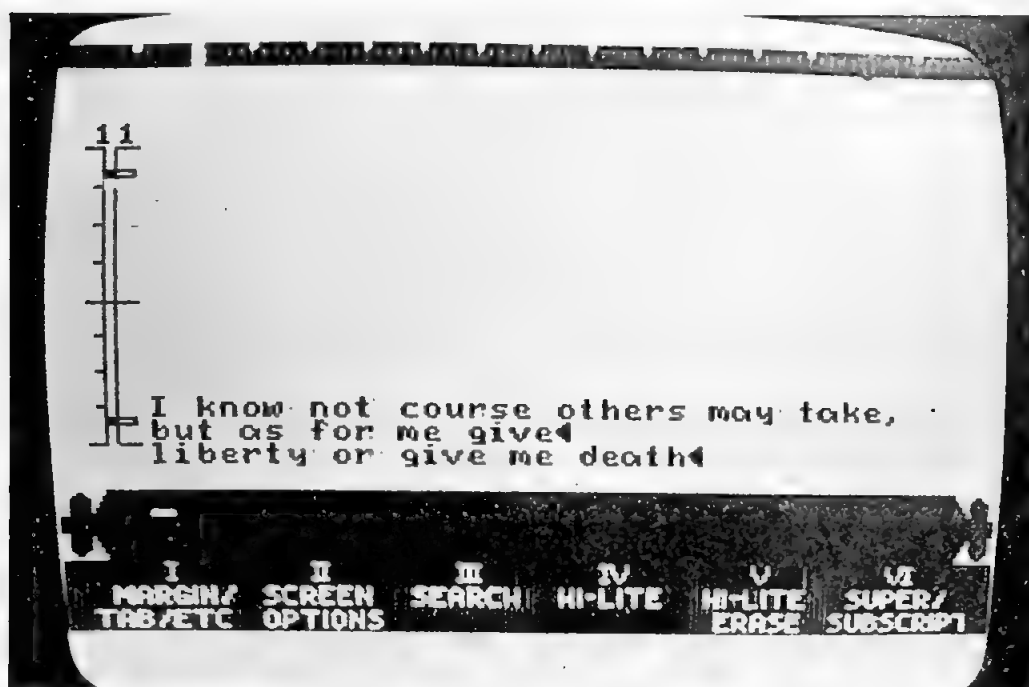


FIGURE 2-2 Initial SmartWriter Screen.

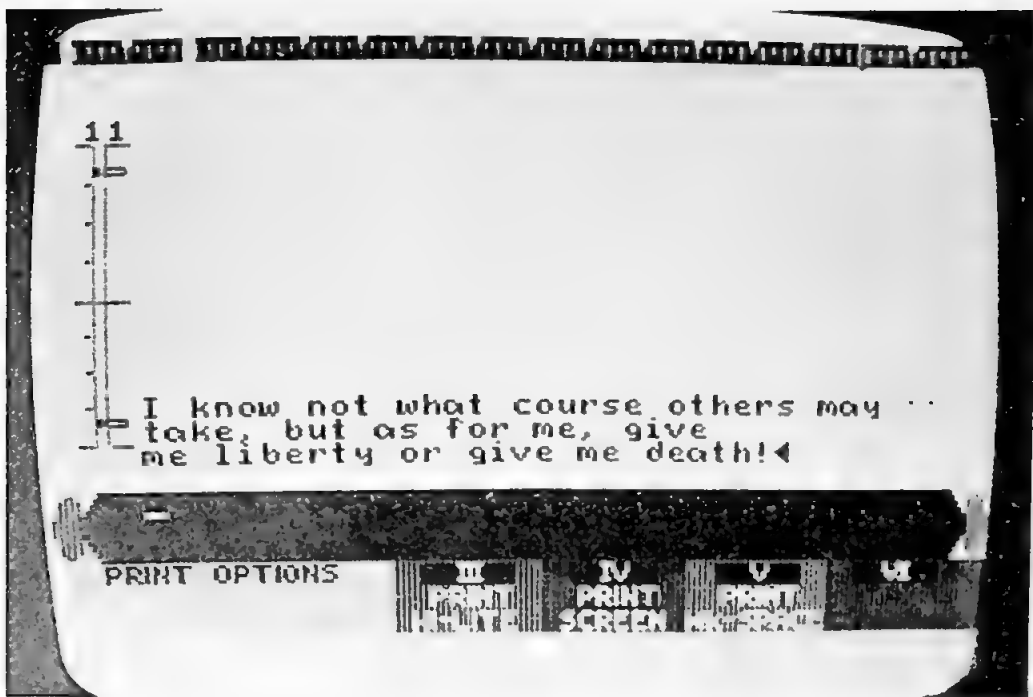


FIGURE 2-3 Print Option Screen.

Press the CLEAR key at the top right-hand corner of your keyboard; you're going to erase all your previous writing. You'll see two boxes on the right-hand side of the screen. Press the dark key labeled VI, CLEAR WK-SPACE. ADAM will ask you, in the message window: CLEAR WORKSPACE: ARE YOU SURE? Yes, you're sure, but thanks for asking! This is how ADAM double-checks to make sure you're not getting rid of something you really wanted to keep. Press VI again and all the text in the document window disappears!

Now type Patrick Henry's remark again, but this time *don't press the RETURN key at all*. Notice that when you reach the end of the line where you had to press the RETURN key after the word "give," ADAM does it automatically for you. You hear a "ding!" and your words continue to appear on the platen. When you've finished the entire sentence, press RETURN again. You'll see an arrow appear after the ! (exclamation point) and all the text will appear above the platen.

Let's print it out. Press the PRINT key (second row from the top on the right, middle key) and you'll see PRINT OPTIONS appear in the

message window. You choose these options by pressing smart key IV, PRINT SCREEN. Another set of options now appears in the message window. For now, just press V, PRINT. ADAM will print out Patrick Henry's statement! Want to print it again? Press the PRINT key, but this time press V, PRINT WORKSPACE. Press V again; in this case you'll only see the same sentence print again, but if you had written more text than you could actually see on the screen, all of it would print out.

Four. Let's see how ADAM works with a longer document. First, CLEAR the workspace again. Do you have a blank screen now? Good. Here's an example of an everyday business letter you can type, but if you'd like to practice this exercise with anything else you've written or would like to write, please feel free to do so. If you make a mistake as you're typing, simply press the BACKSPACE key. It works just like the backspace key on a typewriter except it erases the letter or letters from the screen (and sounds like Pac-Man munching energy pills!). Here's the letter; before you type the return address, press the RETURN key two or three times. This inserts a few blank spaces at the top of the paper. You'll see an > everywhere you need to press RETURN, but the arrows won't appear in your final printout. They are only to show you where you inserted a RETURN. It's a good idea always to insert at least one RETURN at the beginning and end of a letter or document to let SmartWriter know you have begun and ended the file.

>

>

>

23 Terrier Place>

Pawtucket, RI 12345>

January 19, 1984>

>

>

Credit Manager>

Lord & Tucker>

200 Park Avenue>

New York, NY 10001>

>

>

Dear Sir:>

>

My husband bought me a robe as a birthday present from your store in Pawtucket. When I opened it, I found a sewing defect and returned it to the store for exchange.>

>

The clerk told me they had no more of these robes and wasn't sure when they would have them again. I asked if she could issue me credit, since it was purchased on our Lord & Tucker charge card. She was unable to do so, since I did not have the original sales slip.>

>

She took my charge card number and the robe and said she would give the information to the credit department, who could check the files and find the purchase, then credit my account.>

>

I just received my latest statement, and the credit does not appear on it. I am enclosing a photocopy of the statement on which the original charge for the robe appeared, plus a copy of this month's statement. Could you please issue the credit to my account?>

>

Thank you very much.>

>

Sincerely,>

>

>

>

Harriett Rosen>

>

>

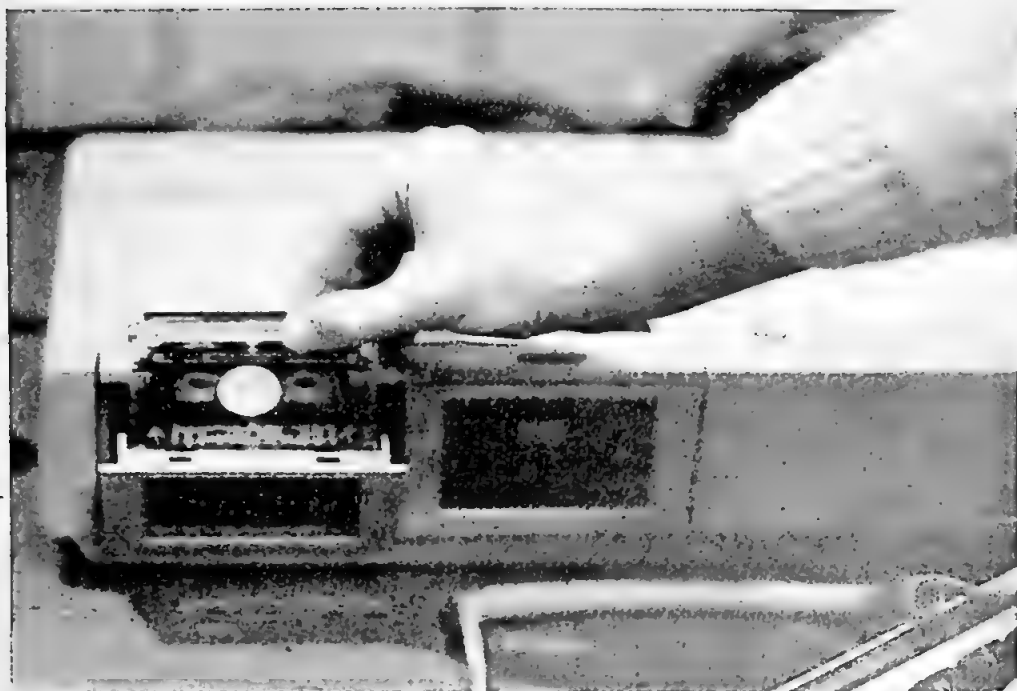


FIGURE 2-4 Proper Tape Insertion.

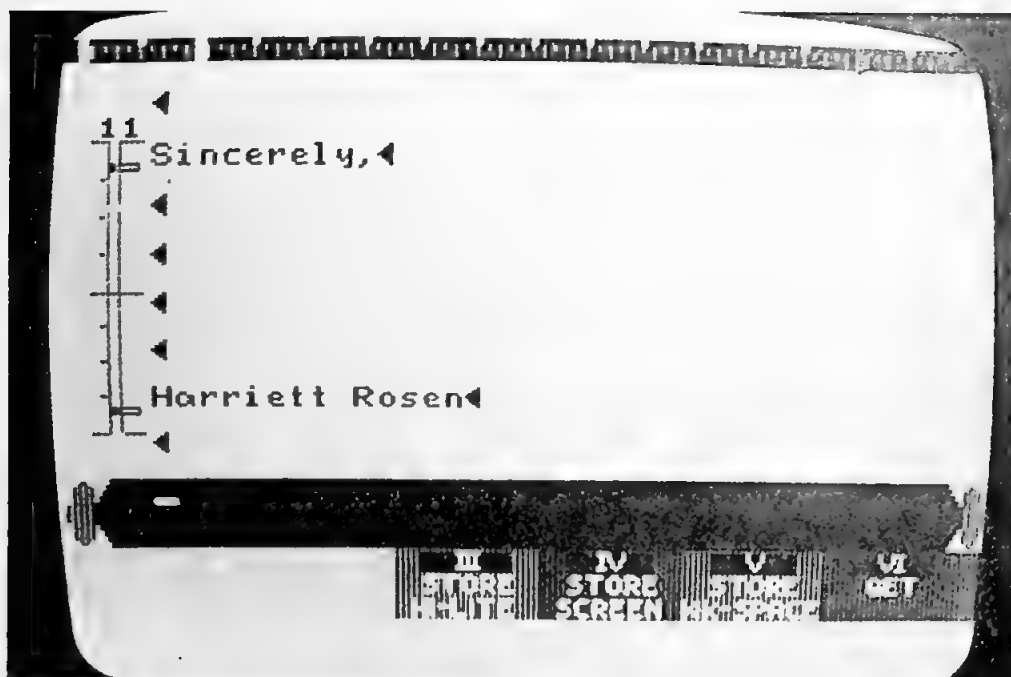


FIGURE 2-5 Store Option Screen.

Five. Finished? Good, now let's store your letter. Press the STORE/GET key; you'll see four options in the message screen. Press V, STORE WORKSPACE. To the left you'll see a blinking message: PLEASE INSERT TAPE OR DISK. Open the tape drive door by pushing the release lock backward, then take a fresh cassette out of its storage box and drop it into the drive *with the side reading "ADAM High Speed Digital Data Pack" facing you and the tape facing down.* Now push the door shut; the blinking message will change to STORE/SELECT DRIVE. If you have only one tape drive, then you'll only see DRIVE. A in box III. Press III and the message screen will read: FOR NEW FILE TYPE FILENAME DRIVE A. A filename can be up to 10 characters long, so call your letter anything you like, such as "credit," "complaint," or "Lord&T." Use names familiar to you, because you can store up to 30 documents in the directory for a cassette, and you'll want to recognize all the different files easily when you read the directory.

Now press V, STORE WORKSPACE. You'll hear the cassette drive start up and begin to spin. Your file is being recorded on the cassette. This takes one or two minutes, so be patient.

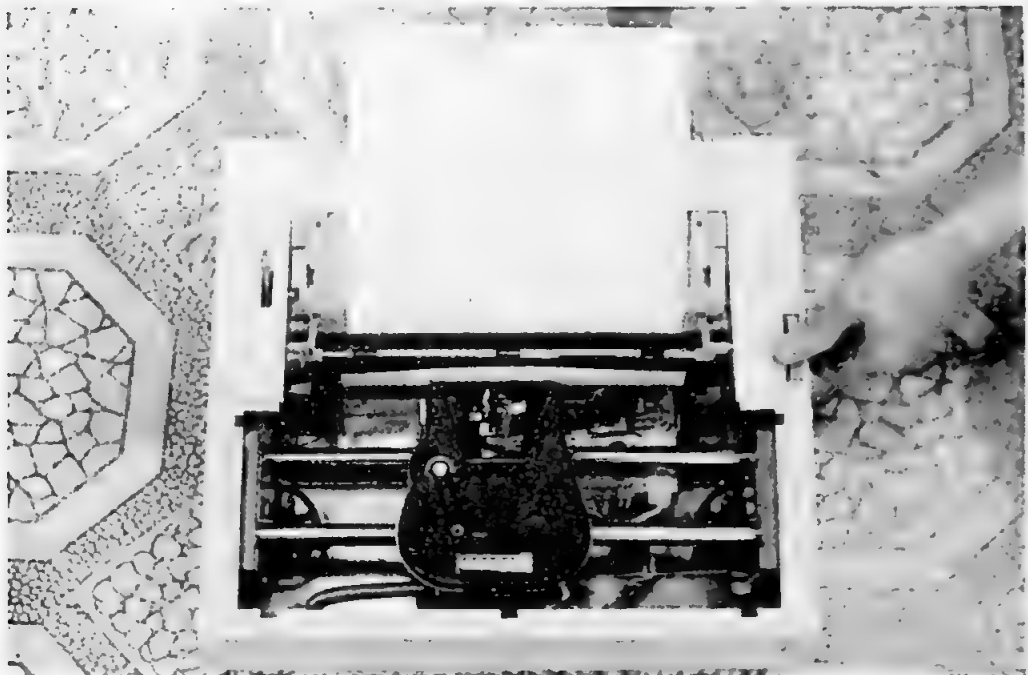


FIGURE 2-6 Proper Paper Feed.

Six. Insert a fresh sheet of paper in the printer. Center it right in the middle, between the two slots or grooves, and roll it in with the thumbwheel on your right. Remember to watch for the print head catching the corner of the paper and slide it to the center if necessary. If the paper rolls in crookedly, pull the paper-bail lever on your left toward you, straighten the paper, and flip the lever back. Position the top edge of the sheet with the top of the clear plastic guide on the print head. With a little practice you'll be able to guide the paper in correctly, and your letters will print neatly and accurately every time.

Now press the PRINT key and follow the instructions in the message window, just as you did before. Within a few moments you'll see your first document printing out! You can't use ADAM while the printer is printing, but if anything happens that makes you want to stop the printing cycle, press V, STOP PRINT. Press the key again to resume printing, and ADAM will pick up right where you left off.

When you're through, press CLEAR once again and clear your workspace. The letter is safe on the data cassette. You might want to practice using SmartWriter by writing a few more one-page documents—a letter, a memo or an essay—until you're comfortable with each step. When you're ready to learn about SmartWriter's other skills and capabilities, turn to Chapter 6.

Getting the Most from Your ADAM Computer

There are many so-called “first” computers, depending on how you define a computer. For example, the abacus, invented around 500 B.C., could be called a computer simply because it counts. The first electronic computer was started by John V. Atanasoff in 1939 at Iowa State College (now University) to help his physics students solve long mathematical calculations.

Other people were making computing devices at the time, but most were used to complete a single task, such as calculating bomb and missile trajectories. It’s hard to believe, but it took almost 10 years for people to realize computers could be used in business!

The microcomputer, or personal computer as we know it today, began its life as the MITS Altair, a hobby kit, in 1975. For a number of years, personal computerists spent most of their time programming their computers to do computer-programming tricks, which caused people to wonder if there was anything at all useful to do with this electronic beast.

Two events changed all that. One was video games, which weren’t really such a new phenomenon; the first video game, Space War, was invented in 1961 at the Massachusetts Institute of Technology. One of the students who loved to play Space War was Nolan Bushnell, who founded Atari! If you ever visit Boston, you can see Space War at the Computer Museum.

The second event was the Apple’s popularity, which inspired many hobbyists to write programs reflecting their varied interests. Almost overnight people realized the personal computer could do a great deal more than just tend to business and finance. It could chart the

stars for astronomers, teach a foreign language, play music, create arts and graphics and store records and information, in interesting and useful ways.

Your ADAM is a state-of-the-art personal computer, unlike any other in its ease of use and the number of things it can do without adding hundreds, even thousands, of dollars worth of extra equipment. ADAM is unique in that it can remain a simple game-playing word processor for you, or it can become a creative tool, a helpful tutor or a sophisticated home manager. It all depends on what you want ADAM to do. What's more, you don't have to spend hours and hours learning this computer to be able to do simple things now; it only takes a few minutes to master new skills. You don't even have to decide all you want to do with ADAM now.

This is not to say it won't take you some time to become accustomed to ADAM, or that you may not spend a lot of time with your ADAM, for you probably will. In fact, computer addiction is a serious subject to psychologists and sociologists. Some people become so involved with their computers that they neglect their friends, families, and jobs. If you find anyone in your family displaying obsessive interest in ADAM, you may want to put limits on the amount of time you permit its use.

Let's take a few minutes to learn about the ADAM system, what it can do now and what it will be capable of doing in the future.

HARDWARE

ADAM is a complete computer as it stands, capable of serving your needs for many years without adding anything to it. But if you become seriously interested in computing, you'll find adding additional equipment enhances your enjoyment and ADAM's usefulness. Think of your stereo or television. At first you were satisfied with just the record player, or tuner, but soon you found you'd like to record your own music, and probably television programs as well. The same is true with your computer, with one exception: adding various components to your computer gives you additional opportunities for your *active participation*, whereas listening to music or watching video is a passive activity. Exploring ways to enhance your life through

computing is what this chapter is all about.

Tape and Disk Drives

ADAM comes with one tape drive, which is sufficient for most things you'll want to do. As you know, SmartWriter is on a chip inside ADAM, and is always there when you turn the power on. SmartBASIC and the Super Game Packs are on data cassettes, and a number of other programs in the future will come on cassette, and later on disk.

But ADAM has a place for a second tape drive; why would you want one? It makes using programs and storing data simpler. Making *backup* copies of your data cassettes is quick and easy with two drives. In the same way you make a carbon copy or photocopy of important letters and documents, anything that's worth saving on a data cassette warrants a backup. Anyone who has worked with a computer will tell you that you only have to lose an important piece of work once to learn the value of making backup copies.

Say you write an essay for a school assignment. Once finished, you STORE it on the cassette. When it's safely stored, you put a fresh cassette in the second tape drive and STORE it again, selecting drive 2. If you write many programs in BASIC, it's a good idea to store them on a data cassette just for your programs instead of on the SmartBASIC cassette. If you decided to use the mailing-list program we give you in Chapter 10, you might want one cassette for the program and another to keep all the names and addresses on.

It's possible to make backup copies using just one tape drive, albeit a little more awkward—you have to keep taking one tape out and putting another in. That's why you may want to consider buying a *disk drive* instead of a second tape drive.

There are several advantages in doing so. One, the disk holds approximately 360 pages of text versus 250. The disk drive operates much faster as well, so it takes far less time to STORE and GET data. After a while you'll become impatient waiting for tapes to load programs and find data; it happens to all of us.

Because they hold so much more data, disks open up a whole new world of software programs for your ADAM. We'll discuss this more later in the section on software.

ADAM's Ribs

Have you ever wondered how a game cartridge works? Inside the black plastic case is a printed circuit board about 1" square with one or two microprocessor chips, just like the Z-80 CPU chip pictured in Chapter 1, mounted on it. In this case, the chip does only one thing: it plays Mr. Do or Space Fury or Turbo. It's not designed to do all the complex things a CPU chip does, but it functions in the same manner.

The instructions for playing the game are permanently etched into that chip. When you plug the cartridge into your ADAM or ColecoVision, the CPU scans or reads it in the same way it scans the data cassette, and you play the game. No ifs, ands or buts. You insert Slither, you play Slither.

We call these ROM chips. If the CPU is ADAM's heart, then ROM is one of ADAM's ribs. ROM stands for Read Only Memory, which means the instructions can only be read and played out one way. You can't add anything to a ROM chip. SmartWriter is on a ROM chip as well; it works just like the game cartridge when you turn ADAM on.

ROM makes sure the computer's work gets done properly. But

ROM makes sure the computer's work gets done properly. But there is another form of memory that makes sure *your* work gets done properly. This is called RAM, for Random Access Memory. Do you remember reading that gobblydegook in the Coleco literature about ADAM's 80K RAM? Well, 80K, which is 81,920 bytes (1K = 1024), is a lot of memory and you should be glad you have so much. It means that ADAM can do a lot more work for you.

RAM, another of ADAM's ribs, is used for two things: holding the program you've loaded into the computer from a cassette for use, and storing the data you're entering into the computer as well. In practical terms it's like having all the paper you need rolling through your typewriter continuously, rather than just a couple of sheets. It's also like the difference between a 45 and a long-playing record. If you had a computer with, say, only 16K of RAM and were writing a program or an essay, it would fill up with words in very short order, and you'd have to stop and STORE it on the cassette every few minutes. You would have to write 50 or 60 pages to fill up 80K.

The reason you should understand RAM is because Coleco offers a memory expansion module that gives you 144K of RAM.

ADAM's Appendages

There are several devices Coleco plans to provide that will make ADAM more useful or allow it to perform other tasks. Most of these devices will be available in 1984; if you are interested in any of them, you can call the Coleco toll-free number, (800) 842-1225, for information. You can order anything Coleco has for sale on the number, too. To help you understand these various devices, which we call *peripherals*, they are described herewith.

Peripherals that make ADAM more useful include:

- *A floppy disk drive*, which gives ADAM the ability (1) to store more data than a data cassette holds, and (2) to run more sophisticated programs, such as those written for the CP/M operating system used on other, more expensive personal computers.

- *The SmartMODEM*, a device that connects your ADAM to the telephone line and opens the world of telecommunications to you. With a modem (modulator-demodulator), you can carry on "conversations" with other telecomputing people all over the country, much as ham radio operators or folks with CB radios do. The difference is that you use your keyboard to send messages, and what the other person writes back appears on your screen. There are thousands of local *computer bulletin boards* all across the country on which users post notices about computers or software for sale, how to join an ADAM user group, how to solve problems you may be having with your computer, and a wide variety of special interests, ranging from politics to education to film to religion. You may also want a subscription to telecomputing *information utilities* like The Source, Delphi or CompuServe.

- If you want to use a color plotter or a dot-matrix printer, you'll need the *RS-232 interface*, a piece of electronic circuitry housed in a box that connects between the modem and the computer, into the modular telephone plug marked ADAMNET on the left side of the memory console. The term *RS-232* refers to an industry standard connection, and *interface* means something that goes between two other things to make them work together. The RS-232 interface may also be used to connect a different printer to ADAM. You may want a dot-matrix printer, or a color plotter, either of which can print out pictures and drawings you make on your computer.

- *The 80-column expansion card*, which allows you to see the full width

of a sheet of paper in SmartWriter. This card must be used with a monitor, described in Chapter 1; a standard television set can only display 40 columns, regardless of the expansion card. If you plan to write a lot, the 80-column card is a must.

- If you write long documents like short stories, magazine articles or book-length manuscripts, you might want the *tractor feed* for your printer. It's a device with gears similar to the tread on a road grader (hence the name tractor) that attaches behind the platen and pulls *continuous-form* paper through the printer. Continuous-form paper comes in a box, usually 2400 or 3600 perforated sheets, with punched holes on either side that slip into the tractor feed. The perforated punched strips zip off, then you simply separate the sheets from one another. You can also buy continuous-form mailing labels and envelopes (see the section on supplies in Chapter 1).

The Ultimate in Viewing: Connecting ADAM to Your VCR

Would you like to tape record the video games you play or the programs you write? It's easy to connect ADAM to a video cassette recorder and record anything that appears on the screen. Simply connect the black cable that comes with your ADAM (for better color, use a shorter cable such as the 6'-long Radio Shack #42-2367) from ADAM's *composite* output jack to the VCR's *video-in* jack. You'll see whatever you're doing on the screen, and the VCR can record it at the same time.

Videotaping offers many interesting ideas for your ADAM. You can study games more carefully and develop better strategies for play. You can record programs you write—shape-table graphics, for instance—so others can watch them later. You can record anything you write in SmartWriter, too: leave notes for the kids when they get home from school, instructions for programs you've written, clues for a mystery party game. The list of things to do is as big as your imagination!

The connection mentioned above provides video only; for video and sound you'll need to use a special cable that plugs into ADAM's large round 5-pin DIN connection. VCRs have a number of different connections, so it's a wise idea to ask your dealer which works best for

ADAM. And you might want to get a video-selector switching box so you only have to hook everything up once.

SOFTWARE

ADAM's future lies not only in the hardware peripherals, but in useful software programs that help you get the most out of your computer. Software is another name for the programs that make ADAM work. It's like sheet music for musicians in an orchestra; they can't play the music unless they know the tune.

Software comes in three different forms for ADAM: tape cassette, micro floppy disk and firmware. SmartBASIC is software on cassette, and SmartWriter is firmware, or software on a chip. (So are the ColecoVision game cartridges.)

Coleco plans to introduce a great many software packages over the next few years, designed for home management, education and enjoyment. As with any computer, it takes a while to design, develop and produce error-free software, so be patient. Take the time now, before there are zillions of software packages to choose from, to learn SmartWriter and SmartBASIC well. These two programs are the most important tools you have, for one helps you become a proficient writer while the other teaches you how to communicate with the computer.

Once you learn BASIC, it is your friend for life. Knowing BASIC is like understanding how to repair a leaky faucet or give your car a tune-up; you're not a master plumber, nor are you an ace auto mechanic, but you're *self-sufficient*. If you want ADAM to do something unique for which there is no program, you can write it yourself. Write games for your children or design an electronic recipe file or a jokes and trivia collection. Create a program that keeps a weather log or that measures energy consumption in your home. The list is as limitless as your needs and imagination. In addition, you'll find many *program listings* in books and magazines, programs other people have written and offer to you. All they cost is the time it takes to copy them into your ADAM; in fact, you'll find program listings elsewhere in the *Companion*. You'll soon see program listings in *ADAM Family Computing* (published by Scholastic, Inc.), to which you should subscribe. And, since SmartBASIC is modeled on Applesoft BASIC, *most* programs in Apple magazines will run on your ADAM. Many useful books and magazines are listed in Appendix C.

Coleco Software

Coleco software falls into three categories:

1. Educational software for the whole family from the age of three through adults, including Dr. Seuss reading and arithmetic, homework helpers, studying for the SATs, or learning about big business, history or political campaigns through simulation games.

2. Programming languages such as LOGO and PILOT, and an advanced assembly language for the Z-80 microprocessor.

3. Household-management programs such as SmartFILER, telecommunications software for use with modem, programs to assist you in SmartWriter, and tutorials for developing specific skills.

Coleco plans to introduce something for everyone with its first four software offerings in 1984. They are as follows:

SmartLOGO, designed for ADAM by its inventor, Seymour Papert of the Massachusetts Institute of Technology. Papert, Dan Bobrow and several other colleagues originally designed LOGO in 1967 as a way to introduce students to programming concepts found in more sophisticated languages. By typing in *words* (*logo* means "word" in Greek) like TO CIRCLE/REPEAT 360/RIGHT/FORWARD/END, instead of abstract mathematical phrases, you could draw a circle. How did LOGO draw the circle? With the Terrapin Turtle, a mechanical device with a pen mounted in its belly that drew on a piece of paper. Later, the people at MIT created a turtle that drew graphics right on the screen.

After the turtle has drawn the circle, you can make it draw another by typing REPEAT. If you position the turtle in just the right place, you can design the seven-ring symbol of the Olympic games. And if you choose a different PENCOLOR for each, they will look just like the real rings!

LOGO not only teaches simple graphics, but can draw solid and spherical geometric shapes, chemical molecules and elements from physics. As you can see, LOGO's not just for kids.

Type Write, a game that teaches you how to type. Typing properly is a skill often neglected in elementary school, but one that becomes increasingly valuable as more and more computer keyboards enter our lives. One estimate is that there will be 80 million computers in the United States by 1987; surely you'll have your fingers on the keys of one at least once or twice a week. Today, if you wanted to learn to

type correctly you'd have to take a six-month course at a business school or community college. Yet the *ADAM Type Write* can teach you at home, without inconvenience or embarrassment, and make it fun at the same time.

Smart WORDBASE/SPELLING CHECKER. It's sad but true that teachers do not place the emphasis on proper spelling they once did. Perhaps it's the price we pay, as a society, for having more engineers and technically skilled people, but there's no reason your letters and documents can't be printed out letter-perfect. This program stores an incredibly large vocabulary, and as it scans your document, it checks each word you've written against those in its memory to see if they're spelled the same. If they aren't it makes a list for you to check and correct.

SmartPicture Processor. ADAM is capable of displaying 16 different colors, which is quite a few. If you could pick up each of these colors as if it were a crayon, think of the beautiful, brightly colored pictures you could draw. Well, you can! Using the hand controller or Super Action Controller, you can choose colors, selecting them by number, then draw using the joystick. If you make a mistake or change your mind, you can simply erase them. Let your imagination run free and think of all the things you could do with electronic crayons—draw your own cartoons, create birthday cards for your friends, or paintings for your parents. But it's just on the screen you say. You can record your work on a VCR and play it on the TV or, if you buy a color plotter and connect it to your RS-232 port, you can print it out and even make color photocopies!

SmartPicture Processor is a *high-resolution graphics* program. If you would like to write your own *low-resolution graphics* program, turn to Chapter 7.

Other Software

Coleco won't be the only company making software for ADAM. Many educational software firms are likely to get on the ADAM software bandwagon, for ADAM is a natural for learning, both at home and in schools. In addition, many people just like you, ADAM owners, have good ideas for games and programs. You may develop a program that teaches how to use LOGO animation, how to create a database manager or a new game. You may want to create a *listing*, which is the

entire printout of the program, and publish it in *ADAM Family Computing*, or you may want to develop it on a data cassette and sell it through classified software ads in the magazine. Some people like the convenience of simply loading the cassette and running the program, while others like to type the lines in themselves. As you'll see later, *ADAM's Companion* offers you programs too long to include in the book in both forms.

A great many games are yet to come. Many are from Coleco, but quite a few are from other software companies, too. You'll find them mentioned in Chapter 11.

You can see that even though ADAM is a complete and useful computer as it now stands, it is also a computer able to grow with your changing needs and interests. For now, please take the time to learn how ADAM works. Then you'll be prepared to understand how the various peripherals work when they're introduced. Develop your competence in SmartWriter and SmartBASIC so you'll have a fundamental knowledge of software that enables you to learn the new programs and languages as they become available.

A Buyer's Guide to ColecoVision Games

While some feel the video-game rose has lost its bloom, *Adam's Companion* feels the petals are just opening. Pong was all the rage in 1977. Next Atari thrilled us with its cartridge games. Coleco wowed us with hand-held "head to head" games, and later wowed us again with ColecoVision. Then last year along came *Dragon's Lair*, the first game to incorporate laser disk technology in an arcade game.

Books and television tell us about experiences others have while we sit passively reading or observing. Video games offer us a chance to participate, to live and relive the thrill of being there as no other medium provides. For a few minutes we transcend our daily lives and enter a rich fantasy world, whether it's fighting interstellar battles, protecting a cherry orchard from alpha monsters and blue chompers, rescuing treasures from incredible underground caverns, or driving in a thrilling grand prix auto race.

Games have been an important, useful tool in the world of computer science for many years. The first Adventure game, written by two students at Stanford in 1971, was used by Digital Equipment Corporation to find bugs, or errors, in its computers. The value of games hasn't been lost on the military establishment either. Guided missile tracking systems are modeled on shoot-'em-up arcade games, and the U.S. Army bought *Battlezone* to train its tank commanders. The Army doesn't call them games, of course: they're referred to as "simulations."

If you think about how far video games have come in the past seven years, from Pong to Buck Rogers Planet of Zoom, you might just agree that video games are only beginning to blossom. This year we'll

likely see an arcade game with a bucket seat that moves and vibrates, complete with stereo sound effects. Before long we may have holographic games with three-dimensional scenes right in our living room. Talk about the next best thing to being there!

COLECOVISION GAMES

ColecoVision games have set the standard for interest as well as arcade-quality play and gorgeous graphics. How good are they? You don't see the cartridges discounted in the stores like the others, and that's a sure sign of their popularity. Parker Brothers, Imagic and Atari, to name a few, now make games for ColecoVision, and you'll probably see a lot more companies join the competition for your video software dollar.

More on Expansion Modules and Accessories

Millions of families have moaned and groaned at the hundreds of dollars they spent on Atari cartridges, only to find the Atari 2600 VCS game system eclipsed by ColecoVision. But things aren't that bad; you can buy the ColecoVision Expansion Module #1 to play your favorite Atari, ActiVision, Imagic and other Atari-compatible games.

Some ColecoVision games can only be played with an add-on expansion module. Turbo, the car race game, requires Expansion Module #2 (they come packaged together). If you have a ColecoVision that you wish to upgrade to an ADAM computer, you add Expansion Module #3, which allows you to play Buck Rogers Planet of Zoom (included) and a number of other Super Game Packs, which are on data cassettes. Super Game Packs offer better graphics, more detailed sound effects, more play levels and additional screens you never saw in the arcade versions. Coleco plans to introduce Zaxxon, Donkey Kong Junior, Subroc, Front Line and others in Super Game Packs. The roller controller, which for some reason is not called an expansion module, is used for playing Slither (included) and more games soon to be announced.

The first extra game-playing gear you buy should probably be the Super Action Controllers. Instead of the flat, square, awkward controllers that come with your Coleco, you get pistol-grip controllers that mold to your hand and have much more responsive joysticks. In

the grip are four triggers for firing and other uses; there is a keypad and a roller ball used for speeding the action. Baseball comes packaged with your Super Action Controllers, which are necessary for Rocky and certain other games yet to be introduced.

EVALUATING THE GAMES

Buying a new game is an expensive proposition, and people often choose a cartridge without much information about it. Each cartridge comes with a booklet covering rules and how to play, and you should read this booklet carefully. But the following evaluations describe what it's like to actually play the game, the sensations and feelings it evokes. Then each is rated, both for playability and viewability. You should be able to get a sense of the game and how you and your family would enjoy playing it. Read on, and see which games you like the best!

RATING SYSTEM FOR GAMES

<i>Playability</i>	<i>Star Rating</i>	<i>Viewability</i>
Trade it in	*	Get a snack
Ho-hum	**	Naps permitted
Good	***	Not bad
Lots of fun	****	Hold onto your chair
Excellent	*****	Grab a controller

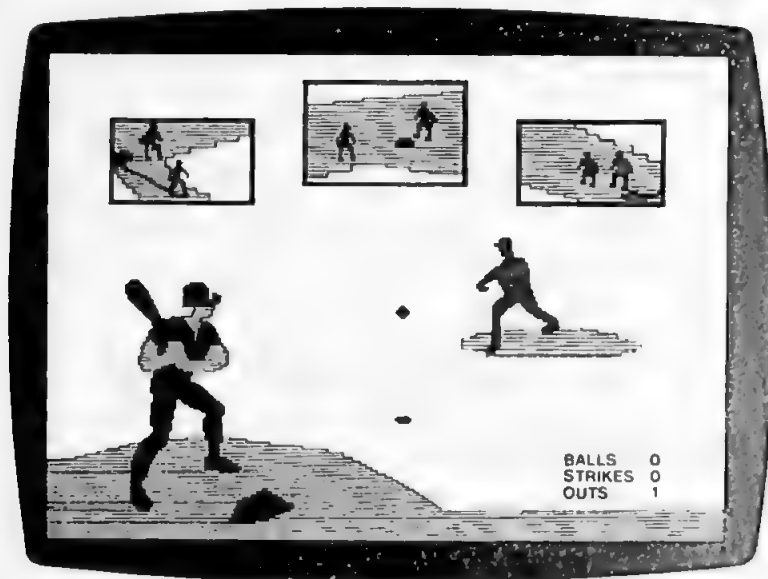
Baseball

If you're a fan of the national pastime, you'll love this one, for no other sports game gives you the same degree of control over play with the Super Action Controllers. If you're pitching, you can choose four pitch speeds on the keypad and select curves, knuckleballs or a straight pitch on the action triggers. Even then, the joystick aims the ball high, low, close or outside. If you're batting, the joystick controls your swing; if you're fielding, it moves the fielders to catch the ball. You can play some real baseball with these controllers.

As the game opens, you see yourself in the foreground, bat in hand at home plate. The pitcher is winding up and throws the ball. If you hit it, spin the roller ball to speed you on your way to first base. The scene changes to a panoramic view of the playing field and you hear

the cheers rise up from the bleachers. If it looks like you can make second, squeeze the blue trigger and spin the roller!

There are many nuances to Baseball—stealing bases, learning to catch fly balls, double plays. It'll require quite a bit of practice to learn how to play well, but that's what a good game is all about, right?



Playability ★★★★★ A game for the very dexterous because you must manipulate the joystick, keypad, triggers and roller ball together.

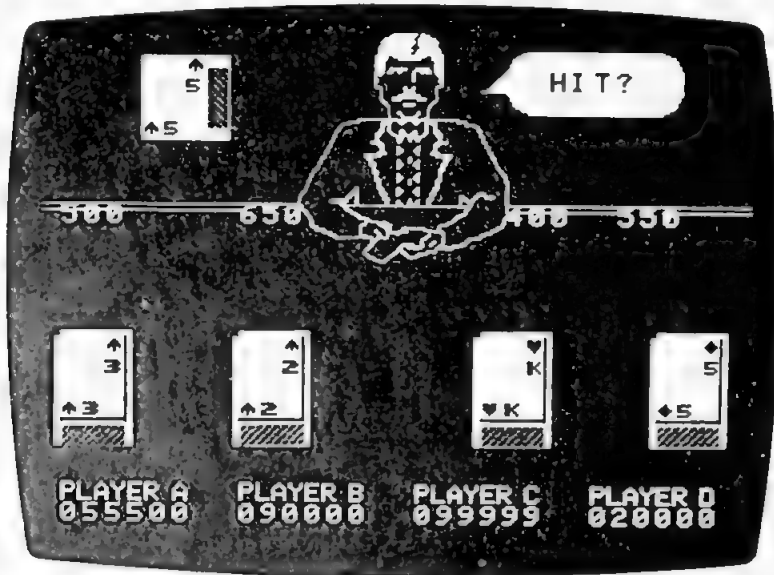
Viewability ★★★ Fun to watch for a while, but the nuances of play are lost unless you have your hands on the controls.

Ken Uston's Blackjack and Poker

Meet Max, the slick, mustachioed Las Vegas card dealer who sits across the green felt tabletop waiting to take your money away from you. The tinkling music makes for a Gay 90s atmosphere, and you're quickly caught up in the game. The more who play—up to four—the better, because everyone can share in the pleasure of beating Max. He smirks whenever he beats you, and displays a toothy grimace when you beat him.

Max plays blackjack, where each player tries to get as close to 21 points as possible, and five-card stud poker. Max shuffles the cards, asks for your bet and then deals with a flourish. You can see him flip

the card and hear it whistle through the air as it lands in front of you. Your fortunes rise and wane, then rise again. Before long you'll have developed quite a relationship with Max while you have some of the best card game fun ever.



Playability ★ ★ ★ ★ ★ Either game is easy to get hooked on: you keep saying, "Just one more hand."

Viewability ★ ★ ★ ★ This one's almost as much fun to watch as it is to play.

Buck Rogers Planet of Zoom

This is the first Super Game Pack and sheer thrills all the way. In play it most closely resembles Luke Skywalker's exploits in the cockpit of the fighter in *Star Wars*. As the game begins you're navigating a channel. You have to reach the Planet of Zoom as quickly as possible, as it's under attack—but so are you! Bouncing tripeds, flying saucers and asteroids bar your way, then suddenly come around from behind and attack. If you're using Super Action Controllers, squeeze the yellow trigger to increase your speed and the red trigger to shoot.

You emerge from the trench into outer space, and the battle rages on until you reach Zoom. The flying saucers are so lifelike that the undersides look different from the top surface. Try your best to survive their onslaught! If you do and reach Zoom's surface, it's time

to show how hot a pilot you are, for you must fly between giant pylons while you fight off more flying saucers. Pull the joystick all the way back to keep as low as possible, and watch where you're going!

You repeat the screens with variations once again, but after the third trip into space you enter incredible adventures on the planet Zoom. You'll encounter the command ship, enter the space warp tunnel and . . . but you should see for yourself what happens next.

The action is fast, the colors have a clarity and resolution unlike anything you've seen on cartridges, and the sounds take you from moment to thrilling moment.



Playability ★ ★ ★ ★ Instead of watching your ship fly, you're in the pilot's seat looking out, so the feeling of movement is incredible. Screens change quickly and challenge your skills and reflexes.

Viewability ★ ★ ★ The action is rather repetitious, but the scenes and colors are outstanding.

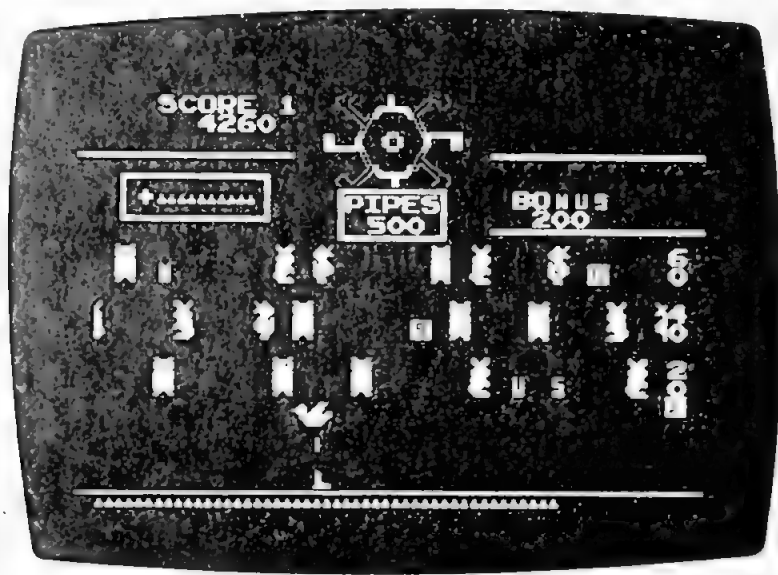
Carnival

Hey! Step right up to the world's greatest electronic shooting gallery! If you've ever been to the carnival, you've probably been tricked or bamboozled into one of the arcades where it looks so easy to win but never is. Well, Carnival is no exception. Your pistol, at the bottom of the screen, moves left or right, and shooting those ducks, rabbits and

owls seems easy enough. Sure enough, every time you hit one you hear "ker-plunk," like hitting a saucepan with a spoon, just like the real thing.

But wait! Those ducks turn into flying geese and try to gobble your bullets! You have to knock them out of the air and score as many points as possible before you run out of ammunition. There are three rows of targets, and the top row garners the highest points. If you can knock off the letters B O N U S in order, a bonus is what you get. Place a shot in the magic box on the left and you gain—or lose—the points displayed. And you must shoot each of the revolving pipes to progress to the next level.

What do you find there? Crazy bears, that's what! Every time you shoot one it rears up and runs in the opposite direction, faster and faster. Can you keep up with it?



Playability ★ ★ ★ ★ ★ The action is quick, but not *too* fast, and there are challenges galore. Loads of fun for younger children.

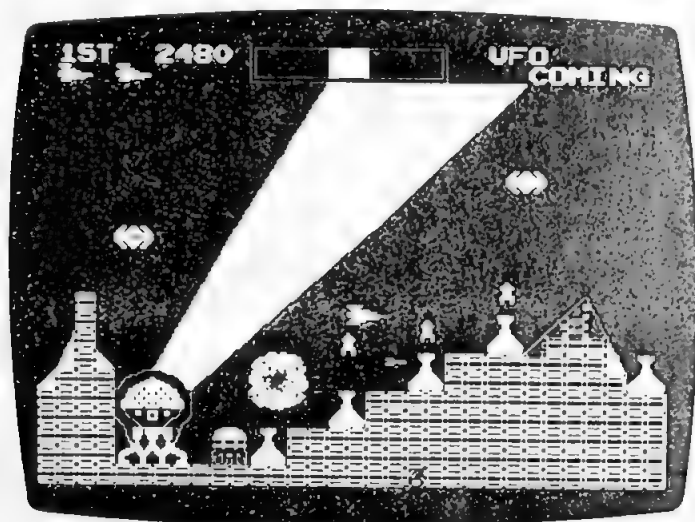
Viewability ★ ★ ★ The bright colors and constant motion make this one fun to watch.

Cosmic Avenger

You are a very unwelcome visitor flying over an alien planet, and these people are really out to get you! They unleash missiles, gunfire,

rockets, shellfire that come after you with incredible speed. A UFO fires at you, dogs you, won't leave you alone. Fly high, fly low, it won't help you escape the firepower from the city below. Your cleverness and piloting skills are essential for survival. Keep track of enemy attack forces on the radar screen, if you have the time. By speeding up you can often outrace your foes, and sometimes you can slow down and duck them as they fly past, then shoot them from behind.

One trigger fires missiles while the other drops bombs, but you must plan your bomb trajectories carefully to hit the target. If you're using Super Action Controllers, firing action is diminished because the first trigger drops bombs instead of firing missiles, a more natural choice. But shoot and bomb your way across the landscape as best you can, and if you're lucky you'll reach the strange, mazelike undersea world. Dodge your way through the seaweed caverns and fend off attacks from submarines, torpedoes and mines. See if you can emerge ready to do battle with the aboveground forces again. Is this one of those days you should have stayed in bed?

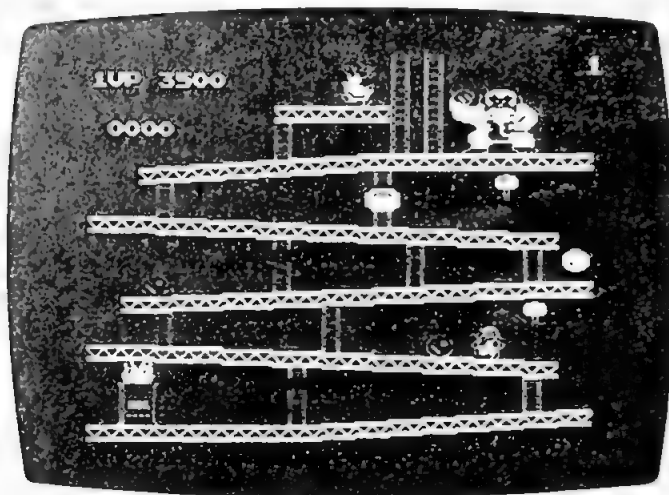


Playability ★ ★ ★ This is a fast, grueling game that is monotonous as well. Who's the cosmic avenger in this game? You don't win, you merely survive.

Viewability ★ ★ The graphics are large, uninteresting graphic blocks, and the unchanging side view of the fighter flying over the terrain or through the water gets boring.

Donkey Kong

Although it's hard to understand just what the name of this game means since there's no donkey, you'll have no trouble figuring out how to play it. It's the age-old story of the guy who rescues his girlfriend from the monster, in this case based on the all-time great movie *King Kong*. Mario the carpenter runs in his inimitable fashion, his little feet making the endearing "blip-blop blip-blop blip-blop" sounds as he goes up the girderworks in the face of barrels, boiling oil, tar men and other troubles. His resources are limited to his jumping ability and grabbing the mallet, which he swings to level anything in his path. Sometimes Mario can stand in just the right place on a ladder to avoid getting barreled over, but eventually he reaches the top and just as he thinks he's reached his sweetheart—you guessed it—we change to another screen.



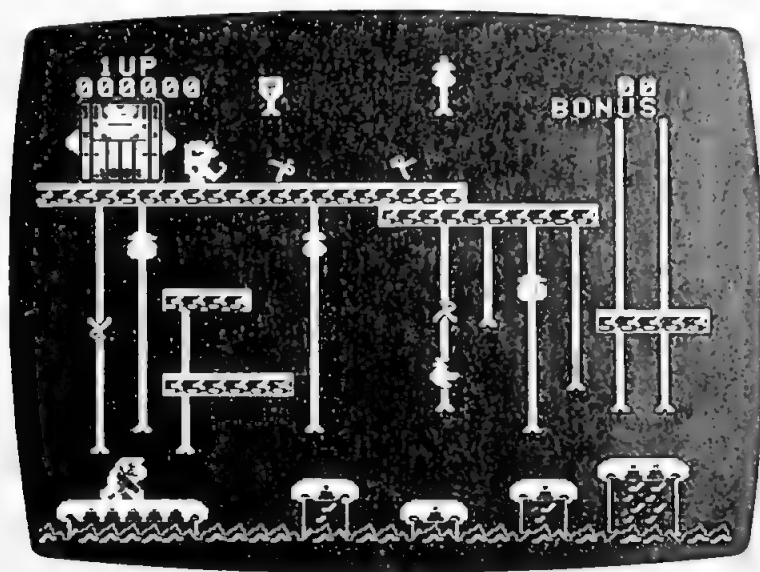
Playability ★ ★ ★ The joystick action is a little stiff at times, and often Mario doesn't go when you'd like him to. Timing is very important in this game.

Viewability ★ ★ ★ ★ This one you can watch for hours, because you want so much for Mario to succeed in his quest. It's root for the good guys!

Donkey Kong Junior

Mario and Donkey Kong are back, but the tables are turned and the gorilla is Mario's prisoner. You're devoted son Donkey Kong Jr. and your mission is to free your father, but many obstacles stand in your way. Sure, you're good at swinging from ropes and vines, but can you dodge the blue and orange crabs that menace your quest? If you're resourceful you can knock them out of commission by dropping ripe fruit on them. But be careful—one false swing or step and it's into the water below!

Once you get to Mario's plateau, run as fast as you can and leap into the air for the silver chalice, which is your key to the second screen. You'll hear a song of triumph that announces that you've made it. But now things get tough. You have to climb the chains that keep your father imprisoned, and you have not only crabs that keep reconnecting the chains, but flying griffins to fight off as well. Will you make it to the next screen? Will you bring your pop home at last?



Playability ★ ★ ★ ★ ★ A delight to play because you use the joystick to control movement and the triggers to jump, and there are so many alternative plays and movements to choose from.

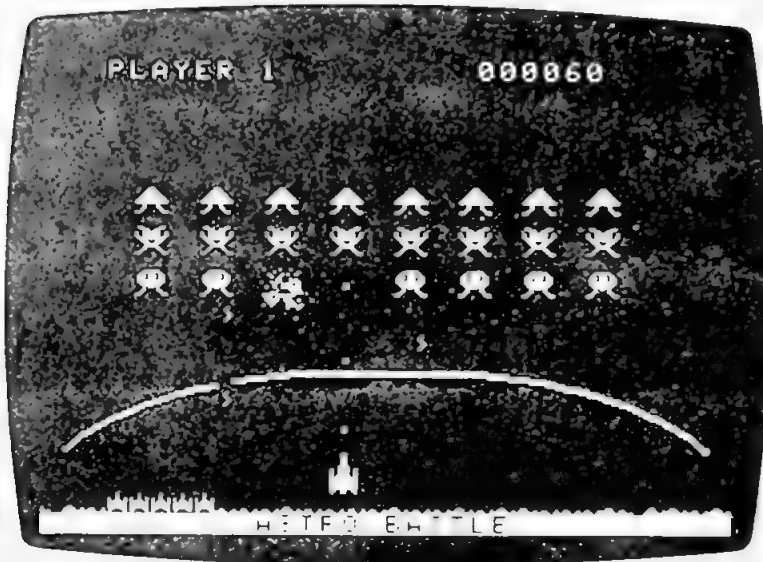
Viewability ★ ★ ★ ★ ★ Everybody loves to watch Donkey Kong Jr. and cheer him on in his antic escapades, throwing fruit, swinging on vines and avoiding the bad guys.

Gorf

You can almost see Gorf, that great omnipotent will hovering somewhere in the cosmos, bent on controlling everything and everyone in the universe by sheer force. And there you are, small, alone, battling it out against unrelenting forces that never let up in their drive to conquer. First come the waves of Space Invader-like critters, trying to destroy your shields. Then the robot ships launch their laser attacks. If you survive that, Gorfian ships come hurling out of a space warp at incredible speed.

It takes some luck, some skill and some strategy to outlast these onslaughts, but the real challenge lies in destroying the Gorfian flagship. You must peck away at its force field until you can plant a bomb in the black heart that is its internal reactor, but meanwhile it's launching intelligent missiles at you with incredible accuracy!

Conquer the flagship and you get promoted from Space Cadet to Space Captain and start all over again. Gorf is interstellar hand-to-hand combat of the first order.



Playability ★ ★ ★ Your ship moves only on the bottom half of the screen, so your movements are limited. Endurance is more of an asset than skill in Gorf, and you must accept the fact that you'll never win, but only roll over to another level.

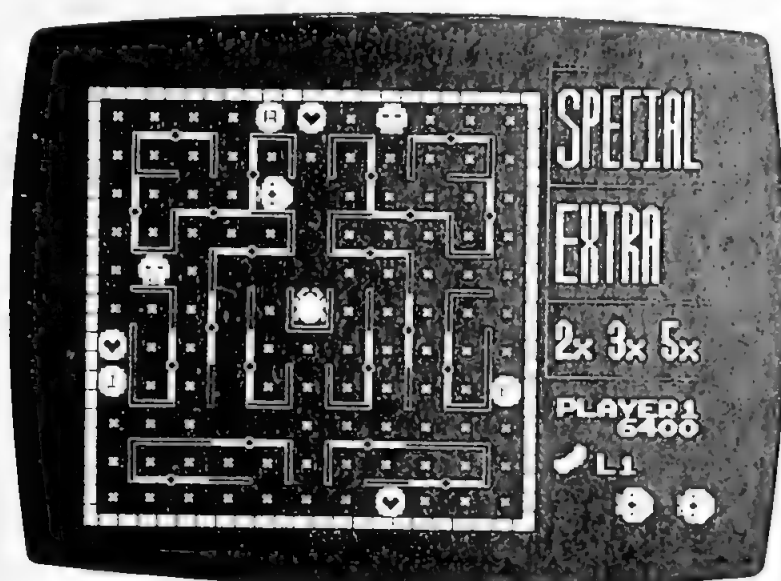
Viewability ★ ★ ★ ★ Brilliant graphics and sound effects of the first

order, especially when you reach another screen. The space warp is beautiful.

Ladybug

Yet another variation on Pac-Man, this game appeals because it isn't predatory. The ladybug peacefully munches dits, vegetables and hearts, avoiding skulls and munching insects. Your goal is to garner points, and the only way to avoid the gobblers is by flipping turnstiles so they can't get at you.

Ladybug is best played with quick flicks of the joystick to negotiate all the quick turns and changing turnstiles. Your goal is to eat all the dots and veggies and, once all the garden pests are out in the maze and after you, the big prize vegetable worth 1000 points appears in the center. Munch it up if you can!



Playability ★ ★ ★ ★ Delightful to play; very little anxiety because your survival depends on how skillfully you *avoid* the insects.

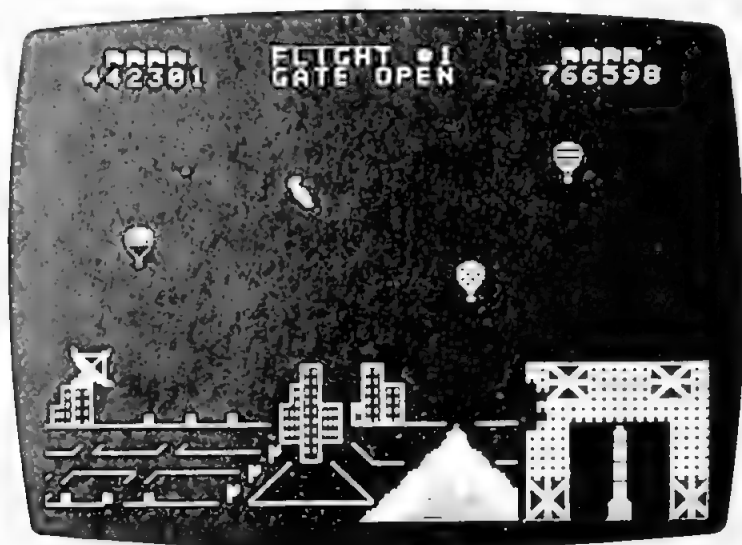
Viewability ★ This is just a simple maze, with little to hold the spectator's interest.

Looping

Get ready for the craziest airplane ride ever! This little ship does everything to avoid going in straight lines, but that's what looping is all about, isn't it?

You're leaving the runway of a large city in a single-engine plane. Pull back on the joystick and you go looping up; push forward and you go looping down. No side-to-side movement is possible. You would be smart to spend your first game or two just practicing flying maneuvers, because it's tricky. Speed helps make your craft more maneuverable; you control this with a joystick button. Large balloons rise to blow you up, but you can shoot them down.

Loop your way into firing position and shoot the rocket on the left side of the screen, then fly away to the right until you reach the entrance to the underground maze of twisting pipes. Now the flying challenge begins in earnest. If you make it, more rooms with more challenges await.



Playability ★ ★ ★ This is a tough game to master, but if learning to be a skillful flyer excites you, you'll love it.

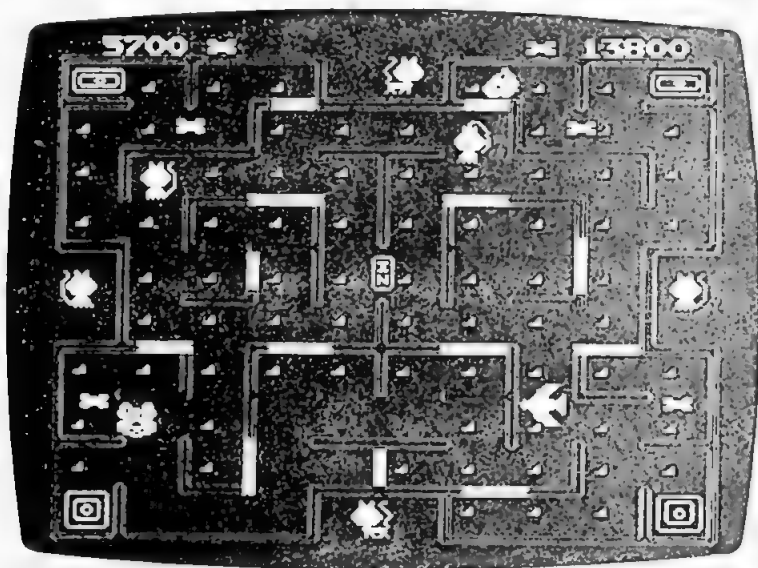
Viewability ★ ★ ★ Very participative, very captivating to watch the little plane looping all around the screen, especially in the maze and beyond.

Mousetrap

If you like Pac-Man, you'll love Mousetrap. You're a cute little mouse, munching cheese nuggets but pursued by a bunch of troublesome cats. The nice thing is that you're not defenseless; press the 5 button on your controller and you turn into a dog, then those cats had better watch out!

Bones are strewn throughout the maze, and you can save them up until you're ready to go on the attack. Of course, you still want to gobble all the cheese and get to the next screen, but the points mount up successively as you bite more cats.

Buttons 1, 2 and 3 open and close maze doors to let you maneuver around; try to get two or three cats near each other, then swoop! Keep your fingers close to the buttons at all times, so you can close a door or transform yourself into a dog. If you play any level above 1, you have to contend with a flying hawk, but you can find refuge from it in one of the corner boxes.



Playability ★★★★★ The Tom-and-Jerry music really keeps you glued to this game with its fast pace. Clearing a screen and earning those extra bonus points entices you into another round of play.

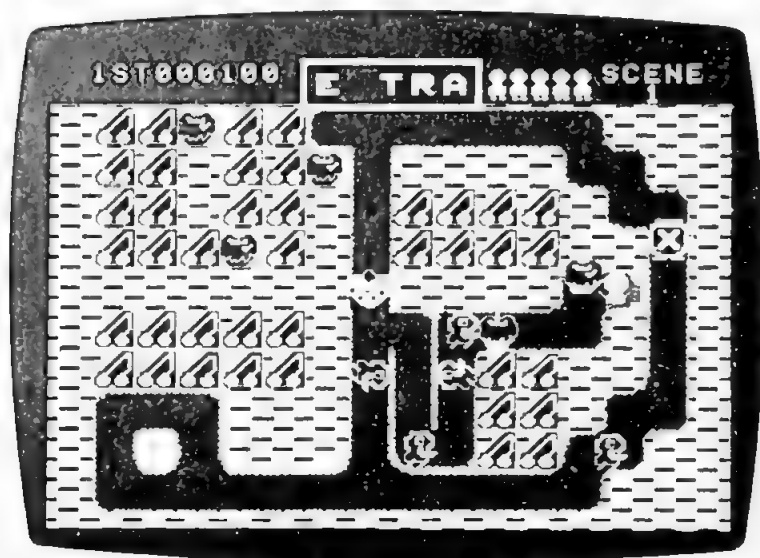
Viewability ★★★★★ Like other maze games, this one doesn't rivet your attention, but it's more fun to watch than most.

Mr. Do!

Poor Mr. Do. All he wants to do is live a pastoral life on his cherry farm, but chomping insects and other meanies make life tough for him. The irony is that they can't go anywhere Mr. Do hasn't cut a path, and Mr. Do can't harvest his cherries without cutting a path! What's a poor farmer to do?

Be as ingenious as possible, that's what. It's not that you've cut a path around the screen, but that you've created a maze to trick and confuse your enemies. And to add even more fun to fooling them, Mr. Do can cut a swath beneath an apple and make it fall on their heads. And if neither of those works, Mr. Do has an incredible Power Ball that works like a boomerang. He unleashes it in the maze and it bounces madly off the walls, eliminates his adversaries and then returns to him.

Pick all the cherries, create a confusing maze, drop apple bombs, battle hordes of Alphamonsters, Badguys, Diggers and Blue Chompers with the Power Ball. Lucky you if you find the diamond and the powers it gives you. It's all in a day's work for a poor cherry farmer, right?



Playability ★★★★★ Outstanding, lively and thrill-packed. The whole family will love Mr. Do.

Viewability ★★★★★ Mr. Do is almost as much fun to watch as it is to play.

Omega Race

This is a wild and woolly version of the classic Asteroids game with a bit of pinball thrown in. You have a weightless starship that bounces off the corners of the screen, which light up every time you hit them. And when you hit them, you bounce! One trigger controls speed, and the joystick allows you to rotate around a square in the middle of the screen, behind which the enemy droid ships hide. One by one they turn into death ships that come after you with a purpose. Fly and bounce yourself into position, then fire away until you clear the screen.

If you have a roller controller, you'll find Omega Race particularly challenging, for the roller takes the place of the joystick in rotating your ship. There are three variations to choose from. One is a square in the center of the screen. The second is two boxes with a tunnel in between for you to sneak through and blast the droids. Three offers astro gates top and bottom, which give you the chance for loads of thrills and the element of surprise.

And if all that weren't enough, you can play head-to-head Omega Race with a friend. Wait till you see *two* ships bouncing off the walls!

Playability ★ ★ ★ Even though the game has many features, it's still a simple shoot-'em-up.

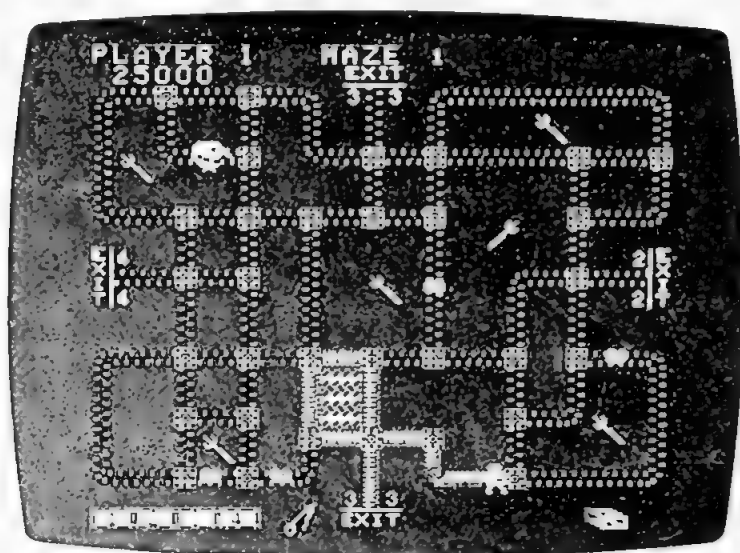
Viewability ★ Pretty boring to watch for more than a minute or two.

Pepper II

Pac-Man comes in many disguises, but this is one of the more interesting ones. Pepper is a little angel spreading goodness throughout the world by zippering up all the boxes in the maze. Some of the rooms have a safety pin, a devil's pitchfork or a magic box inside, which give you a bonus when they're zipped. Unfortunately, Pepper faces the same problem we all do, that evil Zipper Rippers and Roaming Eyes go around undoing all Pepper's goodness. As long as Pepper is zipping boxes and capturing treasures, you can munch the meanies, but watch for the screen color changes!

There are exits on all four sides of the maze that send you into other mazes. You can use these as escape hatches in two ways. You can go on to another maze and zip away for a while until the bad guys catch up with you. Or you can jump out, wait a second or two and jump back in. The bad guys have chased you into the next maze and

you have a few moments alone, but unfortunately, all the rooms that weren't completely zippered up are unzipped again.



Playability ★ ★ ★ ★ ★ Utterly delightful. The Alfred Hitchcock theme music makes this a fun and funny game.

Viewability ★ ★ ★ A bit more interesting than most maze games, but only for short sessions.

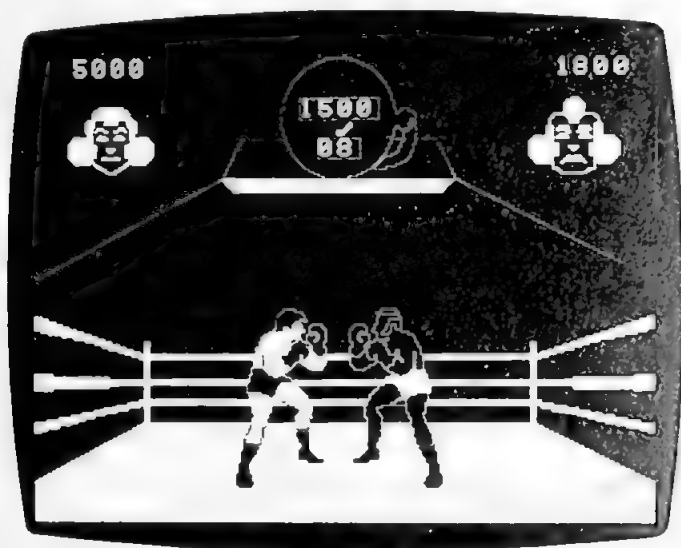
Rocky

Here's your chance to beat up the kid who stole your girlfriend without risking a bruised knuckle or a black eye. Rocky and Clubber slug it out in a boxing ring that looks almost three dimensional. You'll need the Super Action Controllers for this one; the joystick controls movements, while the triggers do the boxing. Yellow is a punch to the head and orange a body blow; the purple one throws your guard up and blue lets you duck a punch.

Rocky and Clubber move and sway around the ring punching, feinting, even getting in clinches. It's fun to watch the referee jump out of their way when they start getting dangerous! In the background you can hear a fan whistle when one of them lands a good punch. Body blows weaken your opponent, while head blows make

him dizzy. It's the two in combination that put him on the mat. You can watch the colors change at the top of the screen to see if you're weakening him.

You can take Rocky's part against the computer, or you can play head-to-head with a friend. How many rounds can you go?



Playability ★★★★★ One of the best. Rocky keeps you on the edge of your seat. The more skill and finesse you develop with triggers, the better the bout.

Viewability ★★★★★ Everyone will enjoy watching Clubber and Rocky battle it out in the ring, as well as the triumphant "Rocky" theme song. The game begins by showing the Italian Stallion depicted in a great work of computer art.

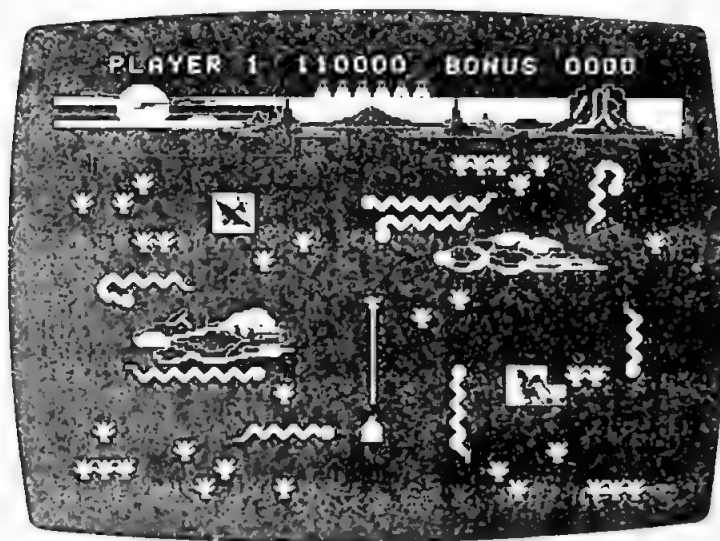
Slither

Crash landing in a stinking desert on this planet was bad enough. Blistering hot by day, cold and desolate at night, not a thing in sight—until this. All of a sudden snakes everywhere, slimy, creepy snakes that split into more and more snakes when you fire on them with your laser.

Then the huge prehistoric birds come swooping in, their ear-piercing radar beeps driving you mad. You'd shoot them down just to shut them up. And those crazy dinosaurs, leaping all over the place.

How do they make the sagebrush grow up instantaneously like that, anyway?

They come at you day and night, day and night. Just when you think you've licked them, here they come again. And now the invisible snakes, the ones where all you can see is their beady little eyes. Snakes gave me nightmares when I was a kid, and now I'm living my worst dreams. Here they come again!



Playability ★★★★★ Only for use with the roller controller, this game keeps you on your toes every minute.

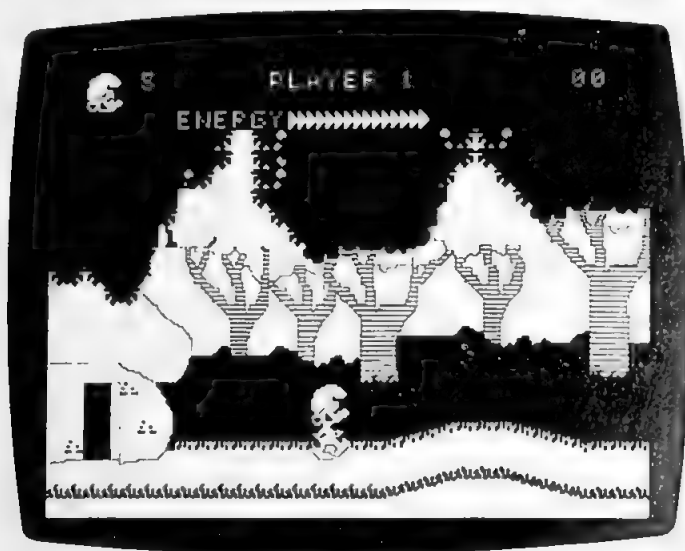
Viewability ★★★★★ It's fun because it's so creepy. Wait until you see the dinosaur eat the ship!

Smurf

This is an adventure game for the younger set. The Smurf sets out on his journey to rescue his darling Smurfette from Gargamel's Castle. Walking through the peaceful forest, he must jump over fences and dodge flying hawks. Then it's on across the fields, hopping up inclines and over bushes. Every jump costs the Smurf some energy, so you must carefully wiggle the joystick for small leaps and large bounds.

Then the little Smurf enters a dark underground chasm, where he must negotiate sharp spires to emerge at the castle itself, filled with dread spiders and other creatures. Since he has no weapons, he must

survive by dodging, jumping and running. Finally, if he's successful, he is reunited with his sweetheart!



Playability ★ ★ ★ ★ This is an easy game, but it requires a subtle hand on the joystick.

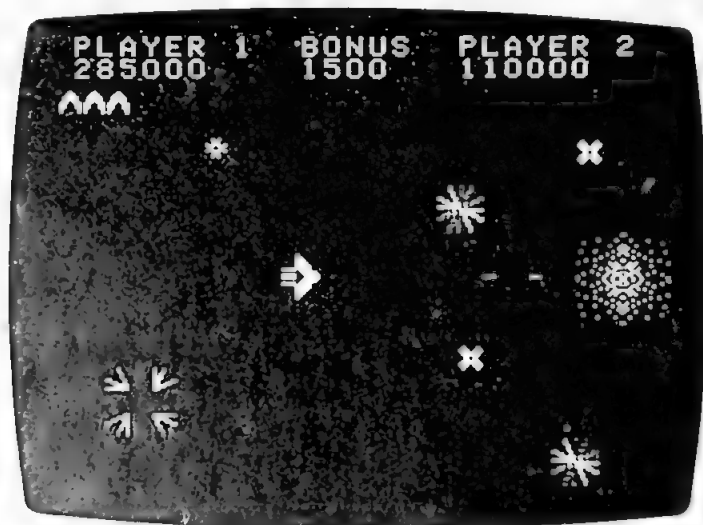
Viewability ★ ★ ★ ★ Charming to watch, the Smurf and the scenes are delightful.

Space Fury

"So! A creature for my amusement! Prepare for battle," says the alien commander, and launches a fleet against you. This is the consummate Asteroids game, full of thrills galore. You begin with a ship in the center of the screen with nose-fire power, the enemy attacking singly but also joining each other by fours to turn into a far more formidable adversary. Squeeze one trigger to fire, the other to maneuver into a better position. Be careful, though; you can race off the screen and reappear in the midst of the enemy ships!

If you blow them all off the screen, you'll get the opportunity to dock with three different mother ships to fight different subsequent rounds. Each docking gives you multidirectional firing power so you can knock the aliens out to the sides and from behind, as well as from the front.

Complete the first round as best you can. Will the alien commander find you a stimulating foe or just amusing? Maybe you'll do even better next time.



Playability ★ ★ ★ ★ ★ Outstanding action for a shoot-'em-up, with plenty of opportunity for skill-shooting and jockeying yourself into better positions.

Viewability ★ ★ ★ ★ Better than asteroids because wave after wave of different ships come after you, and converge into more formidable ships in the process.

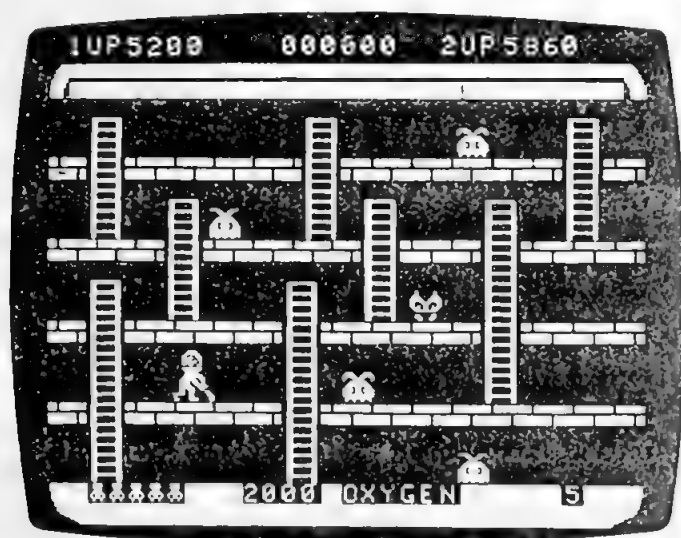
Space Panic

Why would anyone get stuck on a strange alien planet in a building full of monsters that want to gobble him up, and try to get rid of them by digging holes in the floors for them to fall through? Who knows, but if it sounds like fun to you, Space Panic is your game. And a panic it is, either because it's so ludicrous or, if you're playing, because you're trying to dig those holes and cover the monsters over before they get you.

This is Mario with a space suit on, and he only has so much oxygen. All that hard work digging holes wears him out, and he has to keep running up and down ladders, digging new holes, trying to bury the insidious Creatures, Bosses, Dons and Space Monsters. You press one

trigger to dig, another to cover the hole over. If you're not fast enough, the critter comes right back up and chomp, chomp, chomp, you're finished. To make matters worse, some of the monsters must fall through more than one of the four floors before they're quelled.

It's actually fun once you get the hang of it, and before long you have a bunch of holes dug and you're scurrying up and down ladders, burying the buggers like crazy. Keep it up, maybe you'll get off this planet yet.



Playability ★★★★★ Good action, not too difficult to learn, and not nearly as frustrating as Donkey Kong.

Viewability ★★★★★ An involving game; everyone will be crying "Watch out from behind!" or "Dig! Dig!" all the time.

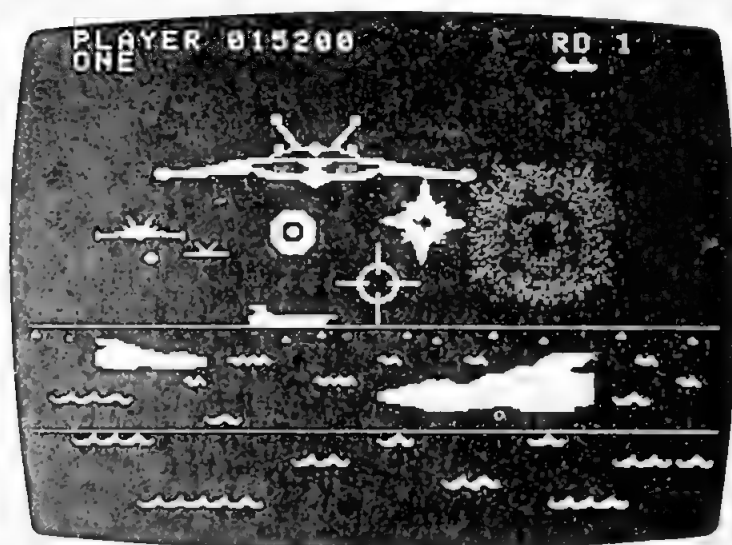
Subroc

You're at the controls of a unique air/sea fighting machine, and looking out of the cockpit at a world full of hostile enemy forces. In the center of the screen, exactly on the water/sky horizon, is the crosshair sight of your weapons system. You can move left and right, up and down, and you'd better to avoid the full-scale attack in progress!

If you dip underwater you can launch torpedoes at passing subs

and ships. Keep an eye peeled for the mystery ship storming across the surface, for it's worth bonus points.

But you mustn't forget the constant barrage from fighters, flying saucers, and missiles. If you ward them off long enough, you'll have a chance to knock the command ship out of the air, but it's tough. There's a moving shield in front of it that blocks your shots, and it's launching missiles at you with a vengeance. Hit the shield and you neutralize it for a moment; fire again and you've got it!



Playability ★ ★ ★ ★ Doesn't require the quick reflexes some of the other shoot-'em-ups do, but it's a good play. It's nice to be in the pilot's seat for a change.

Viewability ★ ★ ★ Fun for a while, but it won't be long until you want to get your hands on the joystick.

Time Pilot

Here's the game that's an aviator's dream come true! You begin at the controls of a World War I biplane, buzzing through the sky like the Red Baron, getting into incredible dogfights. Watch the enemies' remaining window, right below the year you're in. When they're all gone, will you have to do battle with the great zeppelin?

Suddenly, the screen flashes and your plane is caught in a time warp! You're propelled into the future, to the year 1940, and now you're fighting in World War II. Try to knock the bombers down for extra points. If you win your battles here, it's on to the 1970s and scraps with helicopters.

The next and most challenging battle is in 1985 with incredibly fast jets and missiles. Win this round and you'll earn your wings!

Playability ★★★★★ Four different scenarios in four different times give this game its edge. It's great fun.

Viewability ★★★★★ As much fun to watch as to play. The light, fluffy clouds and incredible maneuverability make Time Pilot exciting.

Turbo

You'll need the Expansion Module #2 for Turbo, but it's worth it. This is the ultimate Le Mans race: you sit at the starting line, watching the lights count down to the green flag. Then you race off with the other cars all around you, from the city through the country, into an



incredible curve with a great brick wall on one side and water on the other that never seems to end. Watch out for rear-ends and side-swipes, and keep a sharp eye peeled for screaming ambulances coming toward you. Through the day and the night it goes, while

you try to avoid the terrible crashes and dangerous oil slicks that impede your way to victory. You have 99 seconds to complete the course and earn another round; if you pass 30 cars, you win bonus points.

The steering wheel is very responsive, and any driver worth his salt will have his right foot through the floorboards the entire trip. Throw the joystick up and take off in low gear; shift back into high and the car takes off like a rocket. Into the mountains and through the snow, it's a thrill a second.

Playability ★ ★ ★ ★ ★ The ultimate in road races in your living room. The sights and sounds are terrific, realistic.

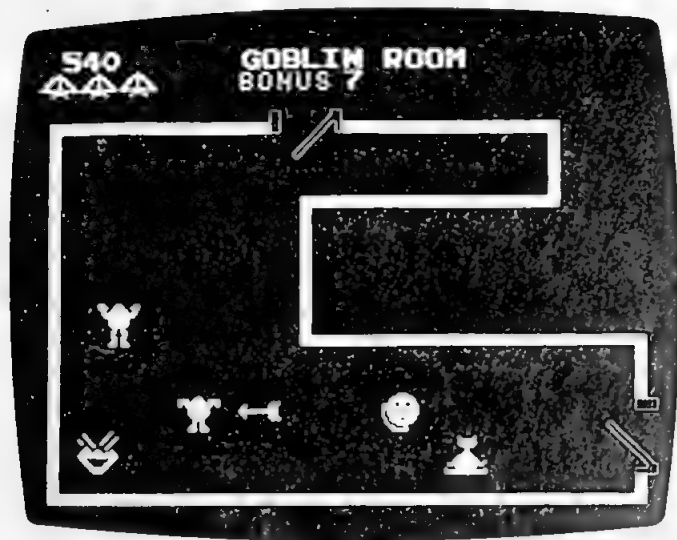
Viewability ★ ★ ★ ★ ★ Turbo puts all the viewers in the room on the edges of their seats. You'll be dying to get your hands on the wheel!

Venture

Ever play the classic Adventure game and wish you had pictures instead of just words to follow? Here's the great escapade! You're Winky, a little critter with a smiling face, off on the great quest to find treasures in hidden vaults and rooms. Each screen has four rooms for you to enter, but they're guarded by the Hallmonsters who'll eat you alive if you come in contact with them. Each room has two doors, and inside each room different monsters and threats lurk. You'll quickly learn that one door is preferable to the other, for if you come in contact with the snakes or goblins you're a goner. Shoot them down or scoot past them, it doesn't matter which, and grab the treasure and head for the door! If you linger too long, a Hallmonster wafts into the room, heads straight for you and winks you out.

Once outside you still must contend with the Hallmonsters, which you can't zap like the critters in the rooms. But wend your way as best you can to another room and grab the treasure inside. Get the treasure from all four rooms and you'll progress to the next level.

Your goal is to fill the grid full of question marks with all the treasures you capture and to emerge from the quest victorious. Good luck—you'll need it!



Playability ★ ★ ★ ★ ★ It's tough, for every room holds new challenges, but you'll find the elation and challenge too much to quit. It may be one of the most intriguing video game challenges you encounter.

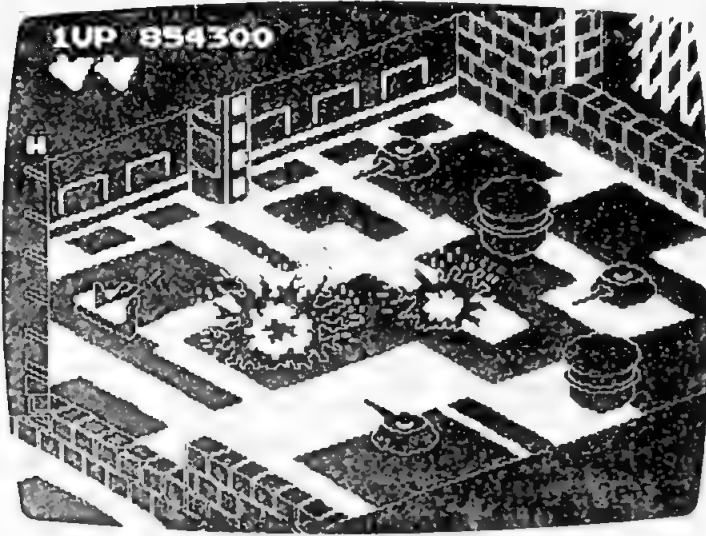
Viewability ★ ★ ★ ★ Fun to watch, but 10 times better to play. This is a solitary venture.

Zaxxon

One of the top arcade games because of its three-dimensional play, Zaxxon lives up to its reputation at home. You're pilot of a starfighter that moves up, down, left, right and in and out, strafing a military base with fuel tanks, stone walls to block your path, heat-seeking rockets and search-and-destroy missiles. Use your ship's shadow and rocket fire to guide you; the lower you fly the more targets you knock out and the safer you are. Every fuel tank you hit replenishes your own fuel supply.

If you succeed in traversing the fortress you find yourself in outer space, battling with the enemy squadron. Keep low and watch for the planes to appear in your gunsights and soon you'll be at the second fortress. Here flying robots shoot at you, but knock them out and you'll meet the incredible Zaxxon, robot warrior par ex-

cellence. If you can hit his missiles and repel him, you win another round.



Playability ★ ★ ★ ★ ★ The most creative shoot-'em-up you can find. Zaxxon is a game you seldom tire of.

Viewability ★ ★ ★ ★ ★ Watching the player try to negotiate in 3-D is a thrill equaled only by playing Zaxxon itself.

How to Design and Write Your Own BASIC Programs

PURPOSE

This chapter

- Explains how to load the SmartBASIC cassette tape.
- Shows how to correct and change BASIC statements without having to retype the whole line.
- Briefly explains how the computer works and how it represents data.
- Demonstrates how you *design* a program so that the actual *writing* is easy.
- Teaches BASIC by example through writing a useful BASIC program to make ADAM a pocket calculator.

You should find nothing mysterious about programming. You do it every day when you write instructions for yourself or someone else to do something. The following instructions may program your daily activities.

Get the newspaper
Bake a cake
Take out the garbage
Pay the bills

A program is nothing more than instructions that tell ADAM how to do something. ADAM cannot take out the garbage, but it does process data and perform arithmetic calculations very fast. With SmartWriter you pressed keys to tell ADAM to perform a process, such as deleting a word or printing a document. With BASIC you tell ADAM what to do by using simple English words like INPUT, PRINT, IF, THEN and GOTO. You use arithmetic symbols such as + (addition), - (subtraction), * (multiply), / (divide) and = (equal sign) to tell BASIC how to perform the desired calculations.

After reading this chapter, you will be able to write your own useful BASIC programs. BASIC statements are instructions to ADAM. Appendix B defines and illustrates all the individual BASIC statements. Please refer to it when a statement is used in this chapter so you understand what it does. Later chapters explaining BASIC graphics commands, how to write a game program, and how to write a mailing-label system will give you enough knowledge and examples to set you off on your own.

Programming takes time and patience to learn, and more time to actually write the program. Debugging—correcting logic errors so that the program works as intended—may be frustrating. Why bother? With all the software available from Coleco and others, why not just buy it?

It isn't worth the time and trouble to write such sophisticated programs as SmartWriter, Donkey Kong Junior or the SmartFiler series. No one person could write all the programs you'll want to use on your ADAM.

Yet buying programs costs money. If you have the time and desire, you will find many programs that you can write. Besides the out-of-pocket dollar advantage, you can custom tailor the program to your own wants and whims. And you may enjoy the creative aspect of programming. While some people have a natural talent for painting or music, you may find you have a talent for computer programming.

Many books and magazines publish BASIC program listings. If a program appears useful to you, simply type it into ADAM, save it on a data cassette, and you have a new, useful program. Often you'll make a typing error when copying a listing or you have to change a command to make the program work on ADAM. This requires some understanding of the BASIC language, which you'll gain in this chapter.

Learning

An old Chinese proverb says, "Give a man a fish and he'll eat for a day, but teach him to fish and he'll eat for a lifetime." *Adam's Companion* gives you a few useful programs, but as it explains how they work, it teaches how BASIC works, a skill that you can use for a lifetime.

Learning requires thought and active involvement. Read the instructions for assembling a model airplane, and a minute later you would have a tough time explaining what you read. But if you actually follow the instructions step by step, then put the airplane together with careful attention, you can explain how you did it to your brother, your sister, or best friend.

If you really want to learn BASIC, the *Companion* will help you every step of the way. Every time a new rule or concept appears, you'll be able to test yourself to see if you've learned and understood. This chapter is designed to make you think and get involved, for learning BASIC requires your serious attention.

If you already have an ADAM, so much the better. Doing is the best way to learn. Let the *Companion* be your guide, but feel free to explore, trying every BASIC statement. Type up the program examples shown here and see for yourself how they work.

If you don't have an ADAM yet, test your learning by filling in the blank words. Take the time to do the suggested exercises. You have spent money to buy this book and time to read these words, so make the investment pay off.

Learning how to program in BASIC means you must use your brain to _____ and become _____ in the learning process. If you cannot fill in the blanks, reread the first sentence in paragraph two of the section called Learning.

LET'S BEGIN

Unlike SmartWriter, SmartBASIC is *not* built in, but comes on a Coleco data cassette called a digital data pack. You load SmartBASIC just as you load the Buck Rogers game pack. Push back the lever above the cassette drive and pop open the door. Place the SmartBASIC tape in the cassette drive with the words *SmartBASIC* on the data pack facing you. Turn the power switch on, then pull the computer RESET switch toward you. As you probably noticed with

Buck Rogers, if there is a data pack in the tape drive when you turn the power on, ADAM has enough smarts to load from the tape rather than execute the built-in SmartWriter program. If ADAM is already turned on, you can change from SmartWriter to BASIC (or any other cassette program) by simply pressing the computer RESET switch. It takes almost two minutes to load SmartBASIC, so be patient. When you see a dash in the upper left-hand corner, ADAM has successfully loaded SmartBASIC. Listen to the whirring sound as the tape rewinds. When it stops, ADAM displays the message WELCOME TO SMARTBASIC and the prompt].

Figure 5-1 (next page) shows BASIC's text screen, which consists of 24 lines each 31 characters wide.

Remove the SmartBASIC cassette tape, put it back in its plastic protective case, and put it safely away in your storage case. While you can store BASIC programs you write on this tape, the *Companion* recommends that you don't. You only have one copy of the BASIC tape and you cannot copy it to another tape. If you damage the tape, you'll have to buy another from Coleco.

WARNING! Never, never place a tape on top of the printer. The magnetic field created by the printer motor will destroy the data recorded magnetically on the tape. Your authors learned this the hard way.

Use the blank tape provided with ADAM to store your BASIC programs. You can store both SmartWriter text files and BASIC program and data files on the same tape if you want to, but you might find it makes more sense to keep BASIC programs and word-processing files on separate tapes.

BASIC

BASIC consists of three parts:

1. The BASIC interpreter program, which is on the SmartBASIC cassette tape. The Z-80 microprocessor executes BASIC using microprocessor instructions. The BASIC program interprets the BASIC statements of your program and performs the commands by executing its own microprocessor subroutines.

2. The language itself uses simple words called commands, like INPUT, PRINT, IF, GOTO and END, and arithmetic symbols (+, -, *, /, ^, =) to instruct the BASIC interpreter program to do some data processing. SmartBASIC was developed so that many Applesoft

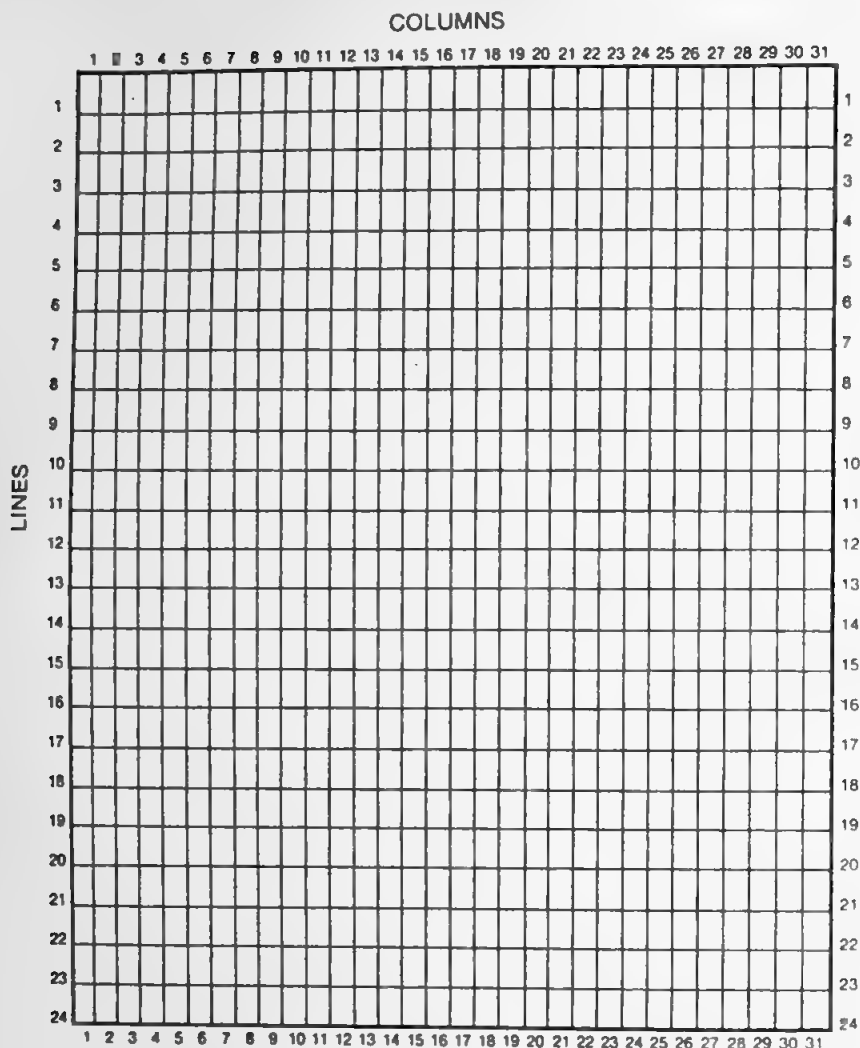


FIGURE 5-1 BASIC Text Mode Screen.

BASIC programs run on ADAM. If you learn SmartBASIC really well, you'll have a working knowledge of the popular Applesoft BASIC.

3. The line editor built into the BASIC interpreter allows you to enter BASIC statements into the computer. It provides several utility functions, and you use simple words like LOAD, LIST, RECOVER and SAVE to tell the BASIC program to perform these functions. As the *Companion* shows you how to write a program, you will get the opportunity to use these functions and see how they work. Normally,

you write a program and store it in memory before RUNning it. However, the line editor allows you to see a BASIC command work immediately without storing it in memory. We call this the *immediate mode* because BASIC executes the command right away, without having to store it first and then run it.

The three parts of BASIC are _____, _____ and _____. Did you write these three parts—*program, language, and line editor*?

When BASIC's Ready

When SmartBASIC displays the prompt

]

it is waiting for you to tell it to do something.

You type NEW

This tells BASIC to clear its memory of all other data and programs because you're ready to start a new program.

ADAM displays]

Even without writing a program you can ask BASIC arithmetic questions and it will give you the answer immediately.

Let's ask ADAM to multiply 32 times 8. Type PRINT, then a space, and the arithmetic you want BASIC to do; then press the RETURN key. Whenever you see RET in the *Companion*, press the RETURN key.

You type PRINT 32 * 8 RET

ADAM displays 256

You type ?32 * 8 RET

ADAM displays 256

The word PRINT and the symbol ? mean the same thing in BASIC. Using ? is just a form of shorthand.

Remember the number 256. You will see it again and again.

The question mark (?) told BASIC to PRINT on the screen the value of the arithmetic expression 32 multiplied by 8. To avoid confusion, BASIC uses the asterisk symbol (*) rather than the small letter x when it multiplies. The table below shows the standard arithmetic symbols in SmartBASIC.

Operator	Operation	Example
+	add	?5+5 RET
-	subtract	?5-5 RET
/	division	?5/5 RET
*	multiplication	?5*5 RET
^	exponentiation	?5^2 RET

BASIC uses other symbols and functions that you will learn about later.

Let's try another problem, adding several numbers, or as we call it in BASIC, an *arithmetic expression*.

You type $?5+3+4$ RET

ADAM displays 12

Did you do it correctly the first time?

Instead of the word PRINT you can substitute the _____ symbol.

The *question mark* (?).

The Order of Calculation

With addition and subtraction, it does not matter which way you add up the numbers. Type the numbers in the last example in reverse order.

You type $?4+3+5$ RET

ADAM displays 12

Whenever your arithmetic expression uses both multiplication or division with addition or subtraction, however, the order becomes important. Consider the arithmetic expression $6/3-2$. If you do the division first, the answer is zero. If you do the subtraction first, the answer is six. Which way will BASIC do the arithmetic? Make it do the calculations and discover for yourself.

You type $?6/3-2$ RET

ADAM displays 0

Without instructions to the contrary, BASIC does _____ before _____. Six divided by three equals two and then subtracting two gives zero. Division preceded subtraction. Figure 5-2 lists the order of precedence for all arithmetic operators.

If you want to tell BASIC to do the subtraction first, you must place opening and closing parentheses around the two numbers or values you want it to compute first. Make BASIC do the subtraction first.

You type $?6/(3-2)$ RET

ADAM displays 6

The _____ made BASIC subtract two from three given that one and any number divided by one stays the same.

Putting an expression in *parentheses* means BASIC will do it first. When an expression has more than one set of parentheses, BASIC evaluates them from left to right.

You type `PRINT (5*3)/(6-1) RET`

ADAM displays 3

BASIC computes _____ and then _____ and divides. Starting from the left, it computes the value of the expression $(5*3)$ and proceeding right computes $(6-1)$. Finally it divides $15/5$ and gets 3.

As you can see, BASIC can evaluate algebraic expressions. You can also enclose one set of parentheses within another set. This is called nesting and, like the single pair of parentheses, it clarifies for you and BASIC the order in which operations are done.

You type `?(7-1)/(5-2)-(8-6) RET`

ADAM displays 0

Remember $6/3-2$? This is the same equation!

To make BASIC first compute $3-2$ you had to put _____ around the expression. Make BASIC compute $(5-2) - (8-6)$ before dividing.

Unless you use parentheses to indicate a different order, SmartBASIC does arithmetic operations in the following order. The operator with the highest precedence number is used first in calculating the value. Refer to Appendix B for a description of the logical operators NOT, AND and OR.

<i>Precedence</i>	<i>Operator</i>	<i>Meaning</i>
Arithmetic Operators		
8	\wedge	Exponentiation
7	$-$	Minus signs used to indicate negative numbers
6	$*$	Multiplication
6	$/$	Division
5	$+$	Addition
5	$-$	Subtraction
Relational Operators		
4	$=$	Equal
4	$<>$ or $><$	Not equal
4	$<$	Less than
4	$<=$ or $=<$	Less than or equal
4	$>=$ or $=>$	Greater than or equal
Logical Operators		
3	NOT	Logical complement
2	AND	Logical AND
1	OR	Logical OR

FIGURE 5-2 Arithmetic Order of Precedence.

You type `PRINT (7-1)/((5-2)-(8-6)) RET`

ADAM displays 6

Now you know how to use *parentheses*.

Immediate Mode

When you entered the `PRINT` commands, BASIC responded immediately with an answer. You have been using BASIC's immediate mode. Actual programming consists of storing a series of commands in BASIC's memory and then telling BASIC to `RUN` them one after the other. For example,

You type `1000 PRINT 6/3 RET`

ADAM displays]

When a line number precedes the command, BASIC just stores the command statement. To execute a stored program you must type the `RUN` command. Do it.

You type `RUN RET`

ADAM displays 2

Instead of giving BASIC one command at a time, a stored program offers you the advantage of writing a series of command statements—a program—and having BASIC execute them all at once. ADAM will store these statements on the cassette tape and can load them when you want BASIC to execute them again.

BASIC operates in two modes: the immediate mode and the stored-program mode. In the _____ mode, when you give a command BASIC immediately executes it. The *immediate* mode differs from the _____ program mode, where commands preceded by a line number are _____ by BASIC in memory. When you type `RUN`, BASIC executes the *stored* statements.

After learning how to correct your typing mistakes, you will write a stored BASIC program.

CORRECTING YOUR TYPING MISTAKES

As you type, ADAM displays the characters you type on the screen, but it also places them into a memory-storage area called an input buffer. ADAM does not process the command you typed until you press the `RETURN` key. If you mistype a character you can use either the `BACKSPACE` key or the left-arrow `←` to move the cursor left. The

character stays on the screen, but BASIC has removed it from the memory-input buffer. Suppose you want to calculate $6/(3 - 2)$ but you mistype it like this:

You type `?6/(3-20)_` don't press return

The underline symbol (`_`) shows the location of the cursor. Now press the left arrow twice to move the cursor back under the zero.

ADAM displays `?6/(3-20)`

BASIC has removed the `0)` from its input buffer even though you still see the two characters on the screen. Type the closing parenthesis and press RETURN and the statement executes correctly.

When you use the BACKSPACE key or `←`, BASIC _____ the characters on the right from its _____. BASIC *removes* the characters from the *input buffer*.

Since the characters stay on the screen, if you move the right arrow `→` you copy the characters back into the input buffer. Suppose you mistyped the left parenthesis like this:

You type `?6/93-2)_`

Press the left arrow four times to position the underline cursor under the 9.

ADAM displays `?6/93-2)`

Now type the left parenthesis, this time holding the shift key.

ADAM displays `?6/(3-2)`

Instead of typing `3-2)` over again, just press the `→` key until the cursor appears to the right of the `)`. As you pass the cursor under the characters on the screen, BASIC copies them into the input buffer. Pressing RETURN always clears the input buffer and when you see the `_`, the buffer is empty.

ADAM displays `?6/(3-2)_`

Don't worry if the cursor moves too far. BASIC ignores extra spaces at the end of the line. By moving the cursor right, you copied the character from the screen into the _____. When you press the RETURN key BASIC processes the *input buffer*.

You can also insert and delete characters on a screen line and then copy the revised line into the input buffer using the right arrow `→`. Suppose you want to insert parentheses.

You type `?6/3-2_`

Use the BACKSPACE or `←` to move the cursor under the 3.

ADAM displays `?6/3-2`

Now hold the CONTROL key down and press the letter N. The screen line shifts right providing space for you to type `(`.

ADAM displays `?6/ (3-2`

From now on the *Companion* will state CTL-N or CTL-letter when it wants you to hold down the CONTROL key and at the same time press a letter key. Now hold down the shift key and type the (.

ADAM displays ?6/(3-2

Press the _____ three times to move the cursor to the right of the two and _____ the characters into the input buffer. Use the *right arrow* to *copy* the characters into the input buffer.

ADAM displays ?6/(3-2

Now type the) and the RETURN key and BASIC prints the value of the _____. BASIC evaluates an *arithmetic expression*.

To delete the parentheses from the screen line use the CTL-O key combination.

You type ?6/(3-2)

Press the ← to move the cursor under the (. Remember, this removes the 3-2) from the input buffer. After deleting the (you have to move the → to copy 3-2 back into the input buffer.

ADAM displays ?6/(3-2)

Now press CTL-O. Magic!

ADAM displays ?6/3-2)

Move the → back to the) copying the 3-2 back into the input buffer and press RET.

To summarize:

CTL-N inserts a space into the screen line, but not the input buffer, at the cursor position.

CTL-O deletes a space from the screen line, not the input buffer.

→ copies the character on the screen under the cursor into the input buffer.

BACKSPACE and ← remove a character from the input buffer.

Correcting Errors After You Hit Return

The up and down arrows move the cursor from line to line but do not affect the input buffer. This means that after you hit RET you can still correct the error in a previous line. Suppose you hit the RET key before typing the closing parenthesis. Don't type the left bracket—the prompt; it is shown here to show that BASIC displays the error line, not directly under the line you typed but shifted one position to the left.

If this were just any old computer, the words SYNTAX ERROR would appear on the screen. But since this is ADAM and Smart-BASIC,

ADAM displays	'Expected
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50
51	51
52	52
53	53
54	54
55	55
56	56
57	57
58	58
59	59
60	60
61	61
62	62
63	63
64	64
65	65
66	66
67	67
68	68
69	69
70	70
71	71
72	72
73	73
74	74
75	75
76	76
77	77
78	78
79	79
80	80
81	81
82	82
83	83
84	84
85	85
86	86
87	87
88	88
89	89
90	90
91	91
92	92
93	93
94	94
95	95
96	96
97	97
98	98
99	99
100	100

You can correct the error by using the up arrow to move the cursor up to either the line you wrote or the copy of the line ADAM displays. To position the cursor to the beginning of the copy line, you must press the **CONTROL** key and left arrow. When you hold the **CONTROL** key down, the left and right arrow keys don't affect the input buffer. Deleting and copying from the screen do not take place.

) Expected

”) Expected

Right!

Remember the _____ and _____ cursor arrows and _____ and _____ arrows with CONTROL key held down don't change the input buffer. You can move the cursor *up* and *down* or with the CONTROL key held down *left* and *right* and *not* change the input buffer. After positioning the cursor, you copy into the buffer with the right arrow alone. The left arrow alone deletes from the input buffer.

Practice on your own. As you write and type BASIC programs you'll learn to use the BASIC line-editing commands effectively, which will save you a lot of retyping time.

HOW THE COMPUTER WORKS

You know enough right now to write a program, but be patient for a few minutes. To understand how to program a computer, you should know a little about how a computer works, what it does and how it stores numbers and text.

Imagine computer memory as one gigantic piece of graph paper, a portion of which you see in Figure 5-3. Each box has a numbered address to give it a unique identity, just as your house or apartment number identifies its location. Each box has eight electronic binary marbles. Binary means two, and each marble is either white or black, representing either one or zero.

An electronic chute connects each box to a control unit. This powerful unit, called a microprocessor, controls the marbles moving from box to box. Given the proper instruction and address, the microprocessor can open the bottom of any box and release a copy of the marbles down the chute and into a location inside the microprocessor. In the example you see a copy of the marbles from location one falling down the chute. They will eventually go inside the

Right. The electronic marbles will go inside the *microprocessor*.

The microprocessor can send a copy of the marbles stored inside it, eight marbles at a time, through a return chute dropping them into a memory box, destroying and taking the place of the marbles that were there. When the microprocessor sends new marbles to a box, the new marbles _____ the old marbles.

If you said *destroy* or *replace*, you understand that when a computer writes new information into a location in memory it destroys the old information.

Why, you might ask, are these marbles sent to these various locations, and why do they destroy the marbles previously lodging in those locations? These locations store the programs on the data cassette tape. The replacement feature allows you to load and play Buck Rogers and then load BASIC so you can write and run BASIC programs. Your BASIC programs will also store data in these boxes.

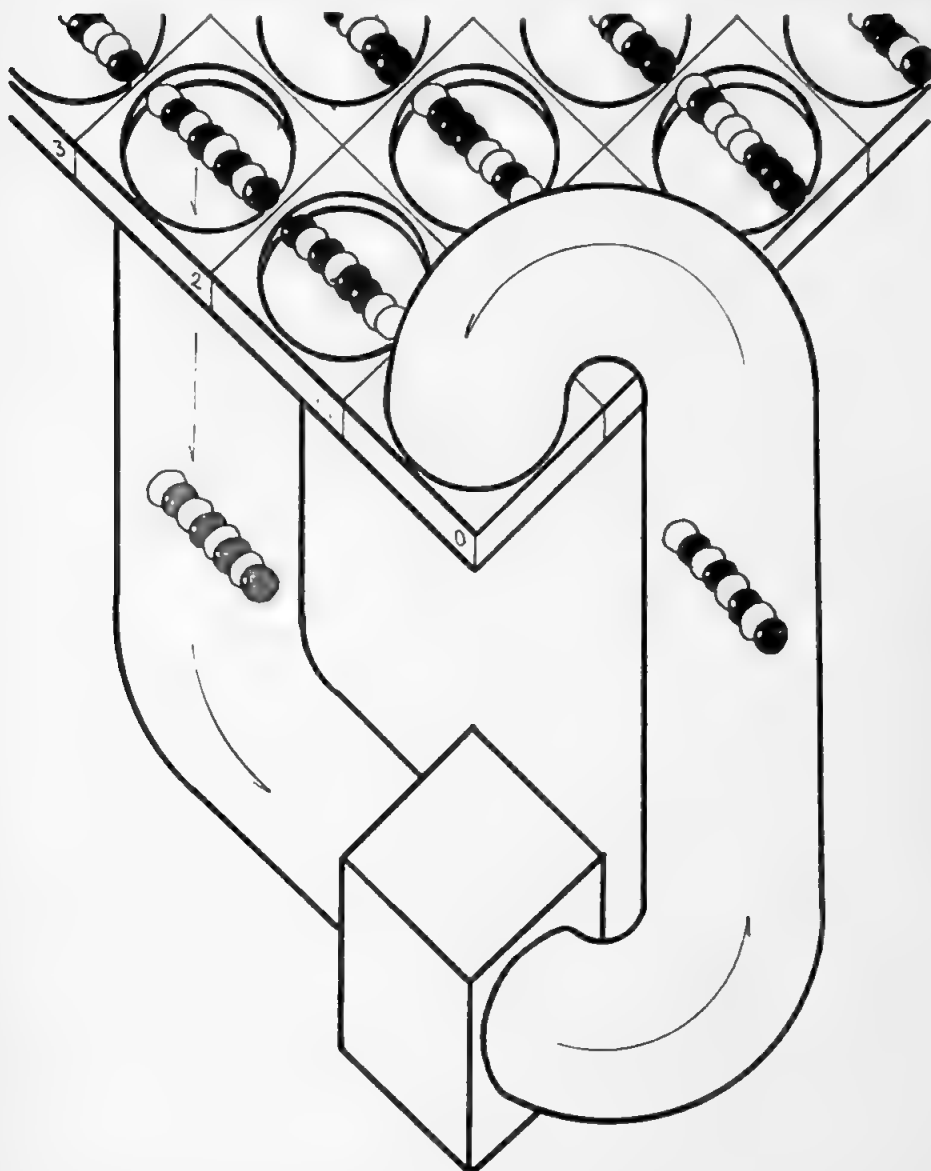


FIGURE 5-3 Marble Memory.

Combinations of Eight Marbles

How many combinations of black and white marbles can you make with eight marbles? Consider just one marble. Remember it has only two colors, white and black representing _____ and zero.

If you can't recall the missing word, try the next number after zero. Computers use only two numbers to count with, *one* and zero.

Now consider two marbles grouped together. The table below shows the value of all possible combinations and the color of each marble. The number inside the parentheses shows the value of the color.

<i>Value</i>	<i>Marble 2</i>	<i>Marble 1</i>
0	black (0)	black (0)
1	black (0)	white (1)
2	white (1)	black (0)
3	white (1)	white (1)

You can group two marbles together in _____ different combinations. Anybody who can count to *four* got that right. For computers the first number is 0; 0, 1, 2, 3, is four.

Adding a marble to the first doubled the power to represent a value from two to four. Observe that the position of the second white marble gives it a value of _____.

Two. The white-black pair represents the value two. Since black has a value zero, white must have a position value of two. The white-white pair represents the value three. The first white has a value one, so the second white must have a position value _____.

Two.

Two marbles have the _____ to represent four values.

Power.

Now add a third marble to the group.

<i>Value</i>	<i>Marble 3</i>	<i>Marble 2</i>	<i>Marble 1</i>
0	black (0)	black (0)	black (0)
1	black (0)	black (0)	white (1)
2	black (0)	white (1)	black (0)
3	black (0)	white (1)	white (1)
4	white (1)	black (0)	black (0)
5	white (1)	black (0)	white (1)
6	white (1)	white (1)	black (0)
7	white (1)	white (1)	white (1)

The third marble has a position value _____ and three marbles have the power to represent _____ values.

Four and eight! Each additional marble has a positional value twice

the previous marble on the left. A decimal number is a 10-colored marble. Each additional marble added to the group of decimal-colored marbles has a positional value 10 times the previous marble added. For example, the left 3 in 33 has a value 30, 10 times the right 3.

Each additional marble doubles the power of the group to represent values. Three 10-colored marbles can represent 1000 values (0-999), three two-colored marbles only 8(0-7).

The table below shows the positional value and power of each of the eight marbles stored in a memory location.

Marbles	8	7	6	5	4	3	2	1
Value	128	64	32	16	8	4	2	1
Power	256	128	64	32	16	8	4	2

From the table, can you figure out how many values eight marbles can represent? _____.

The *Companion* suggested earlier in this chapter that you remember the number 256. You can have BASIC calculate the power of the eighth marble raising the two colors of a marble to the eighth power. In BASIC, the *exponential symbol* ^ raises a number to a power.

You type PRINT 2^8

ADAM displays 256

DATA PROCESSING

By now, you've probably figured out that computers like ADAM do arithmetic using binary numbers, just 0 and 1. People find it easier to use decimal numbers, 0 through 9, to perform arithmetic and data processing. For computers, data processing consists of moving eight marbles, each marble called a bit and all eight a byte, into and out of memory and from location to location in memory. The computer gets or *reads* a byte or character into memory from the keyboard. It sends or writes it to the video screen and daisy-wheel printer.

Each marble represents a _____ and eight marbles are called a _____. A marble has one of two colors and a *bit* has one of two values—one and zero. A *byte* has eight bits and is stored at one memory location. The eight marbles in a byte might end up being a

letter or number or another symbol on your screen or printer depending on its values.

Data processing may seem a little complex to you now, but remember, it's simply instructions. You never would have learned to coordinate your body and brain to drive a car, play basketball or bake a cake without orderly, precise instructions. You cannot program your ADAM to do what you want unless you give it precise BASIC instructions. Now that you have an idea how those instructions work, let's write a program and watch ADAM do some data processing.

YOUR FIRST BASIC PROGRAM

Before you begin writing a program, you must decide what job or process you want the program to do. Will it be an electronic date book? Do you want to draw pictures? As you think about it, list the steps in the process, describing what happens in each step.

I want to draw a picture of a clown.

1. I draw his nose red and his hair green.
2. I color on him a pink suit and big white shoes.
3. He jumps up and turns cartwheels.
4. Every fourth time he jumps, he falls down.

In programming we draw a flowchart that shows how one step follows sequentially to the next. Look at Figure 5-4.

The first program the *Companion* teaches you is the pocket calculator. While you may use BASIC's immediate mode to do many calculations, as we've been doing in this chapter, you may find the pocket calculator program useful, because it has data-identifying features that the immediate mode lacks. More important, if you understand how to write the pocket calculator program, you can use it as a model to write any other program, because other programs listed in the *Companion*, although longer, have much in common with the pocket calculator process and program.

You may be wondering about the difference between a program and a process. A process does something, like adding up numbers on the calculator, while a program consists of the statements that tell the computer how to perform the process. Typing a letter is a _____ and SmartWriter is a word-processing _____. To perform the

Process

Initialization

Input

Decision

Calculation

Output

Termination

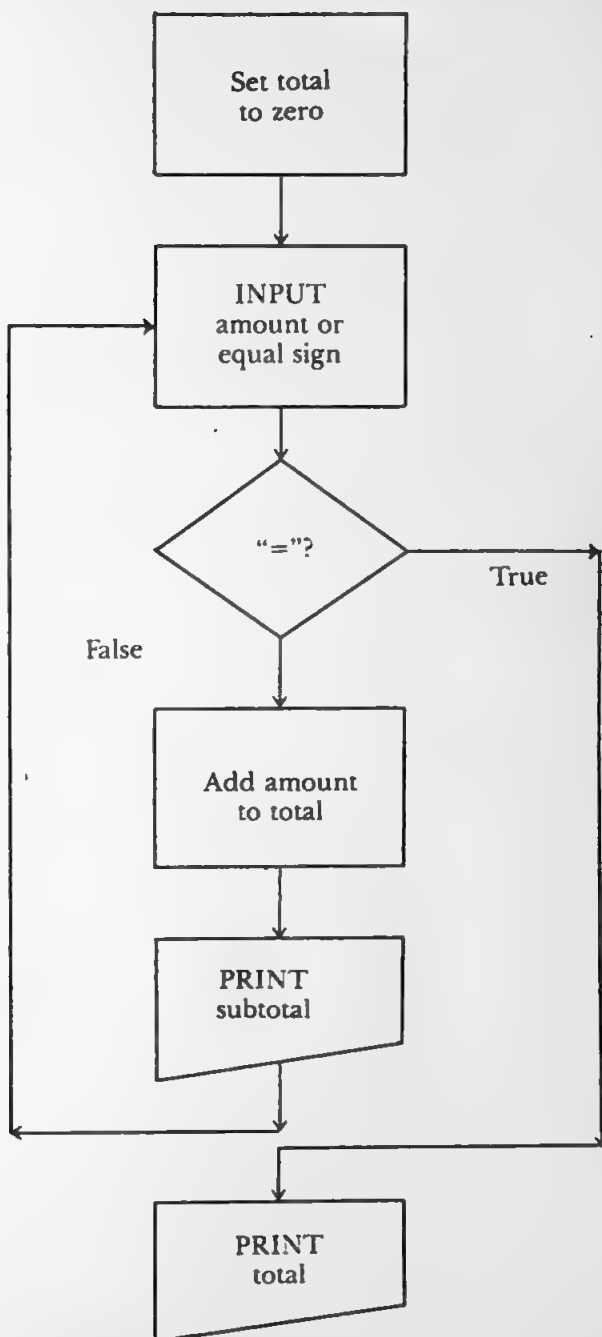


FIGURE 5-4 Flowchart of Calculator Program.

process of typing a letter, you use the SmartWriter *program*.

After you have decided what process you want to program, you must _____ the process to see how it works. You must *study* the process and _____ the steps. *List* the steps and draw a flowchart as shown in Figure 5-4.

The following lists the steps in the process of adding numbers on a calculator. The capitalized words identify commands and names used by the BASIC program.

Step Description

1. Turn on the calculator. The display shows that the TOTAL has been initialized to 0.
2. INPUT or enter the AMOUNT or number by pressing the number-pad keys.
3. To add the AMOUNT, press the operator (+) button.
4. Press either an operator (+) or equal sign (=) to signal the end of the AMOUNT. Now the display shows that the AMOUNT has been added to the TOTAL.
5. IF you press an operator (another +), you GOTO step 2 and INPUT another AMOUNT.
6. IF you press the equal sign (=), the calculator PRINTs or displays the TOTAL.

The flowchart in Figure 5-4 shows that the entire process of adding up numbers contains six fundamental steps or *subprocesses*. Subprocesses then make up a process, which the *program* performs.

1. Initialization: The process of turning on the calculator or pressing CLEAR to set the TOTAL amount to zero.
2. Input: The process of keying in the AMOUNT into the calculator.
3. Calculation: The process of adding the AMOUNT to the TOTAL.
4. Output: The process of displaying the TOTAL.
5. Decision: The process of testing if the equal sign was pressed to check if the entire addition process was completed.
6. Termination: The process of displaying the final TOTAL.

When you study a process, keep in mind that it always consists of these six subprocesses. You can combine these subprocesses in an infinite variety of ways to create any program.

Close your eyes and try to think of the six types of subprocesses just described. If you can't remember them don't worry, because the *Companion* will mention them again as you learn how to design other programs in the subsequent chapters.

Program Building Blocks

Figure 5-5 (next page) shows you the three logical ways to build a program by combining the six fundamental processes above. You can combine them:

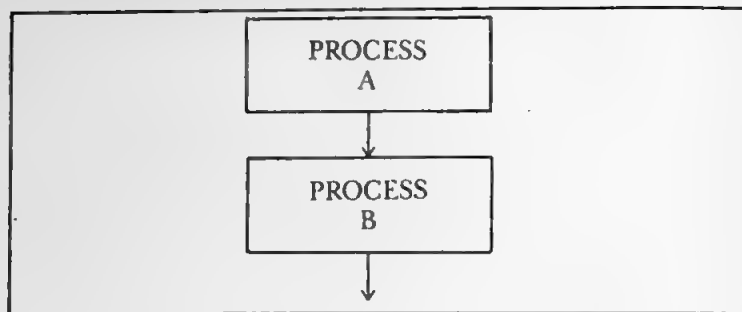
- One after another in sequence like statements in a BASIC program. For example, the statement $TOTAL=5+3$.
- As alternatives for a condition, doing one process if the condition is true and something else if the condition is false. For example, displaying a final total when the equal sign is pressed, otherwise adding the number entered.
- By repeating the same process until a condition becomes true, like adding until the equal sign is pressed.

When you design your program, think how each process relates to the others and to the whole process. Combine the processes in a logical order. Many times the input subprocess must come before the output subprocess. You can combine them *in sequence*, *as alternatives for a condition*, or *by repeating the same process until a condition becomes true*. Figure 5-4 shows how the pocket calculator program will combine the initialization, input, decision, calculation, output, termination in order.

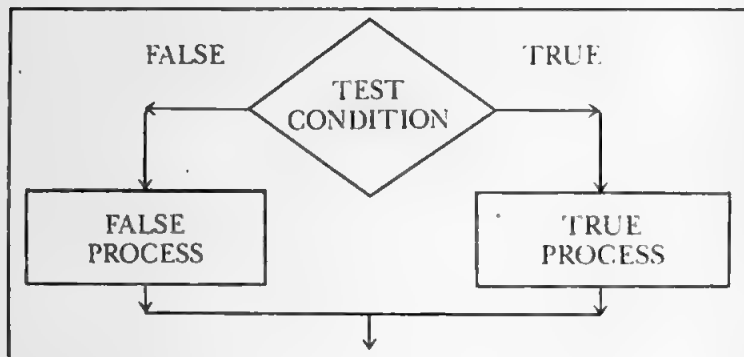
THE POCKET CALCULATOR PROGRAM

To shorten the program for illustration purposes, our pocket calculator program will only do addition. Once you understand the addition program, we'll explain how to modify it to make it do subtraction, multiplication and division.

Sequential
Process



Conditional
Process



Repetitive
Process

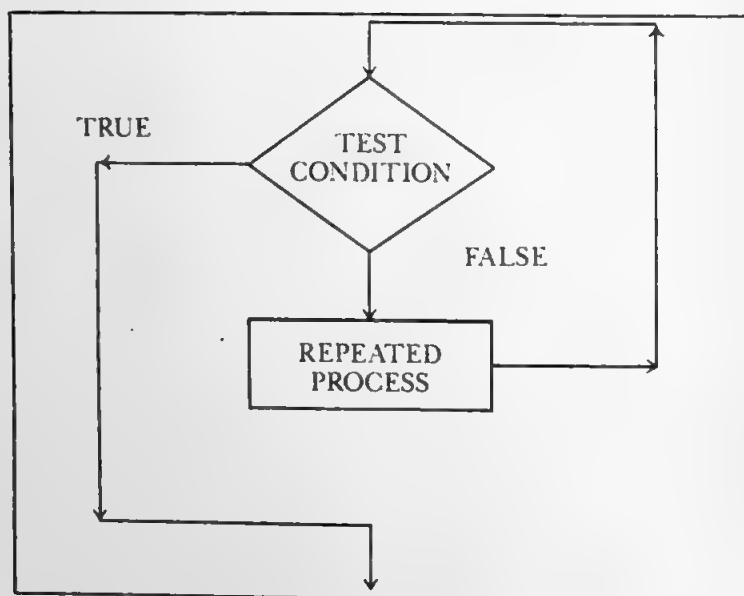


FIGURE 5-5 Flowchart of Program Building Blocks.

Writing the Program

Follow these steps to write the pocket calculator program.

STEP 1 STARTING WITH A CLEAN SHEET

You type	NEW	
<u>ITEM</u>		<u>DESCRIPTION</u>
NEW		Tells BASIC you want to write a new program, and clears its memory of any old program. Make it a practice always to type NEW before starting a new program, otherwise you may mix up old program instructions with your new ones.

STEP 2 IDENTIFY THE PROGRAM

You type	10 REM Calculator	
<u>ITEM</u>		<u>DESCRIPTION</u>
10		The line number identifies the statement so both you and BASIC can refer to it. If you want to LIST or DELeTe a line—you would LIST it to see if it was correct, and DELeTe it if you nò longer need it—you would type LIST 10 or DEL 10. Figure 5-6 (on pp. 85 & 86) summarizes and shows examples of these line-editor commands. Program instructions are executed in line-number order.
REM		The BASIC REMarks command. REMarks are notes to yourself or future programmers that tell you what you did and why. REM commands do not execute—they are for the information of programmers like you and me.
Calculator		The name of the program. It's always a good idea to give your program a name, just like a story or report, so when you look at a printed listing you know what

program you're looking at. Later you will SAVE the program on tape cassette with this name. The *Companion* recommends that in every program you write, you put a REMark command at line number 10 to identify the program.

STEP 3 INITIALIZATION

You type 1000 TOTAL=0

<u>ITEM</u>	<u>DESCRIPTION</u>
<u>1000</u>	Line numbers range from 0 to a maximum of 65535. Leave gaps in line numbers, like 1000, 2000, 3000, so you have room to add additional statements later in the correct sequence.

TOTAL	The name you give to a place, a location in memory, allocated by BASIC, to store a decimal number; called a decimal variable because you can store any decimal number in it. For example, 100000, 123.4 or .000000789. Use a meaningful name, like TOTAL, which names the place where the program will add the sum of numbers entered.
-------	--

=	The equal sign (=) tells BASIC to assign or move the value on the right, in this case zero, to the variable on the left, TOTAL. A zero gets moved into TOTAL, just as it does when you press the <i>clear</i> button on a pocket calculator.
---	--

0	The decimal value zero (0) is a decimal constant, which always represents the same value, as opposed to a variable whose value can change. After turning the calculator on or pressing <i>clear</i> , this number appears in the pocket calculator TOTAL window.
---	--

<i>Command</i>	<i>Description and Examples</i>	
NEW	Clears old program from memory.	
LOAD	Loads a program from tape cassette into memory. Example: LOAD <i>calculator</i>	
SAVE	Saves a program by copying it from memory to tape. Example: SAVE <i>calculator</i>	
LIST	Lists all or part of program in memory. Example: LIST Lists all statements LIST 1000 Lists statement line 1000 LIST 1000-3000 Lists statement lines 1000 through 3000	
RUN	Executes the program in memory or loads the program from the tape cassette and executes it. Example: RUN Executes the program in memory RUN 1000 Executes the program in memory beginning at statement number 1000 RUN <i>calculator</i> Loads the CALCULATOR program from tape cassette into memory and executes it	
DEL	Deletes lines from the program in memory. Example: DEL 1000 Deletes line number 1000 DEL 1000-3000 Deletes line numbers 1000 through 3000	
DELETE	Deletes a file from the cassette directory. Example: DELETE <i>calculator</i> Deletes the CALCULATOR program from cassette tape	
CATALOG	Lists all programs in tape cassette directory. Example: CATALOG	

RECOVER	Recovers a backup file. Example: RECOVER <i>calculator</i>	Marks the backup copy of the program as the current copy; works only if there is no current copy
INIT	Initializes tape directory and places volume name in directory. Example: INIT BASICPROGM	Destroys directory; new directory has volume name of BASICPROGM

Appendix B provides a detailed description of each of these commands.

FIGURE 5-6 BASIC Line Editor Commands.

STEP 4 INPUT

You type	2000 INPUT "ENTER AMOUNT=";AMOUNT\$
<u>ITEM</u> 2000	<u>DESCRIPTION</u> Line number 2000 leaves numbers 1001 through 1999 unused so you have room to add statements later. Small programs often grow into large programs.
INPUT	The BASIC data input command. It asks you, the operator, to enter data via the keyboard.
"ENTER AMOUNT="	BASIC displays these words on the screen, so you know what data to enter. BASIC displays on the screen anything you put in quotation marks. Quotes identify ENTER AMOUNT= as a character string constant, so BASIC knows it is not a variable name or BASIC command. A character string is any group of letters or numbers in a row; if they are stored in a variable they can change, if a constant they don't. In this case it's a constant

—BASIC will always show ENTER AMOUNT=

The semicolon (;) is a required separator.

AMOUNT\$

Variable names must begin with a letter, and BASIC pays attention to only the first two characters. AMS and AMOUNT\$ are both the same to BASIC, but the *Companion* recommends using longer names because they are more meaningful to you than short two-character names.

\$

The dollar sign (\$) tells BASIC that AMOUNT\$ is a character string variable where you will store characters, like "123." Without the dollar sign, BASIC thinks you want to store a decimal number, as TOTAL stored zero. BASIC stores data differently in a string variable than in a decimal variable. A string variable stores up to 255 characters and expands and contracts in size, depending on the characters assigned to it. It takes more room to store "124.45" than "12."

A decimal variable, like TOTAL, stores a decimal number in a binary form designed for arithmetic calculations. While its value changes, the amount of space needed to store the number is the same. It takes the same space to store 124.45 and 12.

TESTING

By RUNning the program as you add statements, you test that what you have written works as you intended. This way, many times you can localize the statement causing the error to what you just wrote.

You type	RUN
ADAM displays	ENTER AMOUNT=
You type	12.5

You type ?amount\$

ADAM displays 12.5

ADAM stored the character string you entered (12.5) into the variable AMOUNT\$. BASIC keeps track of the location of the data. All you have to remember is the name.

STEP 5 DECISION

You type 3000 IF AMOUNT\$="=" THEN GOTO 7000

<u>ITEM</u>	<u>DESCRIPTION</u>
3000	The next line number.
IF	A comparison test command. You use it to compare two values to see if the test is true. Here, you test if the operator pressed the equal sign key indicating that adding is finished and now the TOTAL should be printed.
AMOUNT\$="="	The test, called a relational expression, compares the two items to determine if the relation is true or false.
AMOUNT\$	The variable being compared to the value on the left.
=	The relational operator, which determines the comparison test you want to make. The equal sign (=) means BASIC makes an equal comparison test.
"="	A character string constant, the symbol =. Did the operator remember to enter an equal sign?
THEN	A required word after which you write the commands that you want to perform if the comparison is true. It must always accompany the IF.
GOTO	A command that tells BASIC the next line number to execute; in this case it skips to 7000. If the comparison is

true, you want to display the final total and end the program.

7000

The line number you want to execute if *amount\$* equals an equal sign (=). For example, later you will write line 7000 to PRINT out the TOTAL.

STEP 6 CALCULATION

You type 4000 TOTAL=TOTAL+VAL (AMOUNT\$)

ITEM
4000

DESCRIPTION
Line number

TOTAL

The decimal variable where BASIC will store the value on the right of the equal sign. Here it will store the sum of TOTAL value and the value of the amount entered by the operator.

=

The assignment operator. An equal sign after a variable name tells BASIC this is an assignment or move command.

TOTAL+VAL
(AMOUNT\$)

The arithmetic expression that tells BASIC how to compute the value to move into TOTAL. Here you want to add the input amount to the total and move the sum back into TOTAL. For example, you enter "12.5," then:

$$\begin{aligned} \text{TOTAL} &= \text{TOTAL} + \text{VAL}(\text{amount\$}) \\ &= 0 + \text{VAL}("12.5") \\ 12.5 &= 0 + 12.5 \end{aligned}$$

TOTAL

One of the items in the arithmetic expression. The first time it's zero because you move a zero into TOTAL at statement 1000.

+

Arithmetic operator. Tells BASIC what arithmetic you want done. Here you want to add, so you use the plus sign.

VAL	A function name. This function converts a string into a decimal number. The VALUE function makes the characters "12.5" into decimal number 12.5, which BASIC can add to TOTAL.
(Required separator character.
AMOUNT\$	The argument of the function. The character string variable or constant that the VAL function changes to a decimal. The "argument" is a variable that the function operates on to produce a value.
)	Required separator. All functions have their arguments enclosed in parentheses.

STEP 7 OUTPUT

You type 5000 PRINT "SUBTOTAL="; TOTAL

<u>ITEM</u>	<u>DESCRIPTION</u>
5000	Line number.

PRINT A command that displays a list of constants and variables on the video screen and, if directed, to the actual printer. You can also use ? instead of the word PRINT. You want to print TOTAL so the operator can see the current sum, just as the calculator shows you the TOTAL after each add operation.

"SUBTOTAL=" A string constant. Follow good programming practice and use a character string to identify the output. Help the next person using your program to use the program and to understand the output. A program helps people, not just gives computers work to do. Someone using this program to add numbers could figure out that the number displayed means a subtotal, but labeling the data makes it

immediately clear that the amount is a subtotal.

;

A separator between items in the list. It means don't leave any spaces between the items displayed. For example, if TOTAL equals 12.5, then BASIC displays
SUBTOTAL=12.5

TOTAL

The decimal variable that contains the current sum of all numbers entered.

Always remember to put a PRINT statement in your program to see what you told BASIC to do. Without the displayed output all the other processing has no benefit to you.

TESTING

Take the time to make sure that the statements you have written so far work as intended. Can you enter a string of numbers, convert them to decimal, add them to TOTAL and PRINT the TOTAL?

You type RUN
ADAM displays ENTER AMOUNT=
You type 12.5
ADAM displays SUBTOTAL=12.5

STEP 8 LOOP BACK AND DO IT AGAIN

You type 6000 GOTO 2000

<u>ITEM</u>	<u>DESCRIPTION</u>
GOTO	A command that changes the sequence of line-number execution. While Looping is a popular Coleco game, <i>looping</i> is also a programming technique to repeat a group of statements. The purpose of the calculator program is to add up a series of numbers entered by the operator. So far you have gone through the calculator processes only once. The GOTO command allows you to loop or circle back to

the input command in line 2000 and add a new number.


2000

The line number of the input statement where you enter another number. The arrow below shows what happens when BASIC executes the GOTO command.

```

10 REM    calculator
1000 total = 0
2000 INPUT "enter amount="; amount$
3000 IF amount$ = "" THEN GOTO 7000
4000 total = total+VAL(amount$)
5000 PRINT "sub total="; total
6000 GOTO 2000

```



STEP 9 TERMINATION

You type 7000 PRINT "TOTAL=";TOTAL

<u>ITEM</u>	<u>DESCRIPTION</u>
7000	Line number
PRINT	To display the final TOTAL line on the screen.
"TOTAL="	A character string to identify the output.
;	List item separator.
TOTAL	The decimal variable containing the final TOTAL.

Congratulations! You have written your first program. After running it to make sure it works, you want to list it on the printer and save it on cassette.

Printing the Program on the Printer

BASIC allows you to have more than one command per statement line. You use the colon symbol (:) to tell BASIC that there is another statement on the line.

To LIST the program on the printer, first make sure that you have

aligned the paper to the top of the page. Then type (in capitals or lowercase):

PR#1:LIST:PR#0

PR#1 tells BASIC to send all output to the printer. LIST displays and prints the program. PR#0 turns off the printer. You may want to use a three-hole punch and then insert the listing in a three-ring binder for safekeeping and organization purposes.

Initializing the Tape

When you get a blank cassette tape you may want to put a name on the electronic directory to identify the data you will store on the tape. Use 1 to 10 characters for the volume name.

You type INIT volumename

You can also use this command to effectively eliminate all files from the tape so you can reuse it.

BEWARE: This command destroys the existing directory of text and program and creates a new empty directory. Without the directory you have no way of reading the data files on a tape.

Displaying the Directory

To display the directory

You type CATALOG

If you have a new tape BASIC will just display the volume name.

Saving Your Program

The *Companion* recommends you save programs with the same name you used in the REM statement on line 10. This way you can identify a listing with a name on a catalog displayed from the cassette's directory. Also, write the name of the program on the index card in the plastic cassette holder. To save the program on cassette, place the cassette in drive 1 on the left (SmartWriter names this drive A).

You type SAVE calculator

If you have a two-drive system and want to save the program on drive 2 on the right (SmartWriter names this drive B),

You type SAVE calculator,D2

The drive 2 setting stays in effect until changed. Now if

You type CATALOG

SmartBASIC will read and display the catalog from drive 2. To save another copy of the program on drive 1,

You type SAVE calculator, D1

If you only have one drive, you have to replace the cassette with the backup tape and repeat the SAVE statement.

By the way, SmartBASIC keeps a backup copy of your program on tape when you save the program a second time (the same is true of SmartWriter text files). If you need to read it, use the RECOVER command after you first DELETE the current version of the program. To RECOVER the previous version or the program VIDEOGAME,

You type CATALOG

This makes sure you have a backup version. Type a lowercase letter *a* next to the name.

You type DELETE VIDEOGAME

You type RECOVER VIDEOGAME

You type LOAD VIDEOGAME

Loading Your Program

When you turn off ADAM, the BASIC language interpreter and your program disappear from memory. You already know how to load BASIC. When you load a program you replace the current program in memory, if any, with the program from cassette. Before loading a program you may want to save the existing program. To load a program,

You type LOAD name

When BASIC finishes loading the program it displays the | prompt.

SUMMARY

The CALCULATOR program has:

- Demonstrated how a few BASIC statements work;
- Provided you with a model for almost all the future programs you will write; and
- Illustrated the point that a process, such as adding numbers, is itself made up of the following subprocesses:

- Initialization
- Input

Test for condition
Calculation
Output
Termination

If you understand the pocket calculator program, you can write almost any BASIC program. You have learned that before you write a program, you must analyze the *process* you want to *program* and break it down into *subprocesses*. Just like the process of putting eggs into a cake mix breaks down into picking up the egg, tapping it against the bowl, and pulling apart the cracked eggshell, so do many seemingly complex processes divide into simple subprocesses. Take the time to figure out the subprocesses, how to combine them and which conditions control their execution. Repeat the procedure until you have reached the level where you can write a subprocess in a dozen or so statements. Then write the program for the subprocess. Test it to make sure it works. Then combine each tested small program to create a larger process until finally you have the main program. Using this approach, given time and the knowledge of how a few more BASIC commands work, you can write any program.

Subtraction, Multiplication, and Division

The chapters on graphics and writing a video game illustrate how to develop large programs. To demonstrate how small programs, like the CALCULATOR program, evolve into bigger programs, the *Companion* will expand the calculation subprocesses to add more features. For example, you can replace

4000 TOTAL=TOTAL+VAL (amount\$)

the statement that performs the calculation process, with a few statements, to make the CALCULATOR program do subtraction, multiplication and division.

How You Do It

With a real pocket calculator, you enter a number and one of the symbols (+, -, *, /) to indicate the arithmetic you want done. You can make the CALCULATOR program work the same way. When the program asks:

ENTER AMOUNT=

you enter the number and the arithmetic symbol. For example, to multiply the TOTAL by 45 you enter:

"45*"

Character String Functions

You need to insert statements in the CALCULATOR program to test for the arithmetic symbol, just as it now tests for the equal sign (=). To do this, you need to select the symbol from the right side of the input string\$ in *amount\$*. Suppose you entered "45*"; the program must select and identify the asterisk (*) to determine that it must do multiplication.

Functions, like the VAL function, return a value. The VAL function took a string argument and returned a decimal value. The character string functions take a string argument and return part of the character string. The contents of the original string remain the same; you just get a copy of part of it. You use the RIGHTS string function to get part of the right side of a character string. Type the following commands to print just the arithmetic symbol, the last and rightmost character in *amount\$*.

You type *amount\$*="45*"

You type ?RIGHTS(*amount\$*,1)

ADAM displays *

The variable *amount\$*, the first argument, is the character string\$ you want part of. The second argument, the value one, tells BASIC to take only one character from the right side. The size of the partial string you get depends on the value of the second argument. The value varies from one up to the length of the string. If you use a value equal to or greater than the length of the string, you get the entire string. Here, *amount\$* has a length of three, so if you use a value three or more, BASIC will give you the entire string. To try it.

You type ?RIGHTS(*amount\$*,5)

ADAM displays 45*

ADAM displayed the entire string because the second argument, five, was greater than the length of the string of characters in *amount\$*. Just ?*amount\$* would have displayed the same.

To get part of the right side of a character string, you use the _____ function. *RIGHT\$!*

The `LEFT$` function works the same way, but returns a partial string from the characters on the left side of a character string. To get the number 45 from the string "45*" now in `amount$`

You type `LEFT$(amount$,2)`

ADAM displays 45

In the pocket calculator program, the length of the `amount$` will vary. One time you may enter "9/" which has a two-character length, and another time you may enter "1234+" which has a five-character length. To get only the number part, from the left side of the input in `amount$`, you need a second argument value one less than the length of the character entered. You don't want the arithmetic symbol on the right side. For example, to get the four characters "1234" from "1234+" you need to subtract one from the length of "1234+" ($5 - 1 = 4$). To get the value for the length of a string, you use the `LEN` function. To print the `LEN`gth of `amount$` that contains "45*"

You type `?LEN(amount$)`

ADAM displays 3

There are only three _____ in `amount$`. There are three characters at the location named `amount$`: 4, 5 and *, or 45*.

Now put divide by 9 in `amount$`.

You type `amount$="9/"`

You type `?LEN(amount$)`

ADAM displays 2

The new contents of `amount$` have a `LEN`gth of 2.

To get the numbers from any size `amount$` string,

You type `?LEFT$(amount$(LEN(amount$)-1))`

ADAM displays 9

BASIC does the arithmetic first, subtracting 1 from 2, making the statement read `LEFT$(amount$,1)`. Then it gets a partial string, one character long, from the left side of `amount$`.

Appendix B explains the `MID$` character string function, but you don't need it for the `CALCULATOR` program.

THE POCKET CALCULATOR PROGRAM

To change the `CALCULATOR` program so it does more than just addition enter the following statements:


```
4000 op$ = RIGHT$(amount$, 1)
4100 size% = LEN(amount$)-1
4150 IF LEN(amount$) < 2 THEN GOTO 4700
4200 amount = VAL(LEFT$(amount$, size%))
4300 IF op$ = "-" THEN total = total-amount: GOTO 5000
4400 IF op$ = "*" THEN total = total*amount: GOTO 5000
4500 IF op$ = "/" THEN total = total/amount: GOTO 5000
4600 IF op$ = "+" THEN total = total+amount: GOTO 5000
4700 PRINT "bad input"
4800 GOTO 2000
```

Statement 4000 assigns the partial string containing the arithmetic symbol into the string variable named *op\$*.

Statement 4100 computes the value of the second argument for the LEFT\$ function in 4200. For example, if you enter "631+", using the LEN function with *amount\$* returns a length of 4 including 1 for the plus sign (+). Subtracting 1 gives the size of "631" which is 3 and is stored in the variable *size%*. This will allow the LEFT\$ function to select the number part of the input.

The percent sign (%) after the variable name *size%* means you store only whole numbers or *integers*, numbers without a decimal fraction, at its location. For example, 45.671 has a decimal fraction .671 and you use a decimal variable to store it, but 3 has no decimal fraction and you can store it as an *integer* variable. The LENGTH of a string will always have a whole number or integer value. Figure 5-7 summarizes BASIC's three variable types—decimal, integer and string.

If *size%* is less than one, the operator won't enter a number or symbol. The CALCULATOR program requires that you always enter both a number and a symbol, like 9/. Entering just a single character, like the number 9 or a symbol like / is wrong. If LEN(*amount\$*) is less than one, BASIC will transfer control to statement 4700, which PRINTS an error message, BAD INPUT, and then transfers execution back to line 2000, the INPUT statement.

Statement 4200 converts the character string into a decimal number. The decimal variable named *amount* is different from the string variable *amount\$*. These name two different locations: one stores a decimal number and the other a character string.

Statements 4300 through 4600 check the variable *op\$* to determine which arithmetic operation to perform. After doing the arithmetic, each statement GOTOs the PRINT subtotal statement. If the operation symbol is not +, -, * or /, the BAD INPUT message gets displayed.

You use a name, called a variable name, to identify the memory location where SmartBASIC stores data and the type of data stored there. The three data types decimal, integer and character string are shown below.

SmartBASIC only recognizes the first two characters to identify the variable. In other words, the name *am* and *amount* refer to the same memory location; SmartBASIC ignores the *ount*. The name must begin with a letter from A-Z. The optional second character can be a letter or number. Valid variable names are:

f

f1

fo

found

Note: both *fo* and *found* refer to the same data

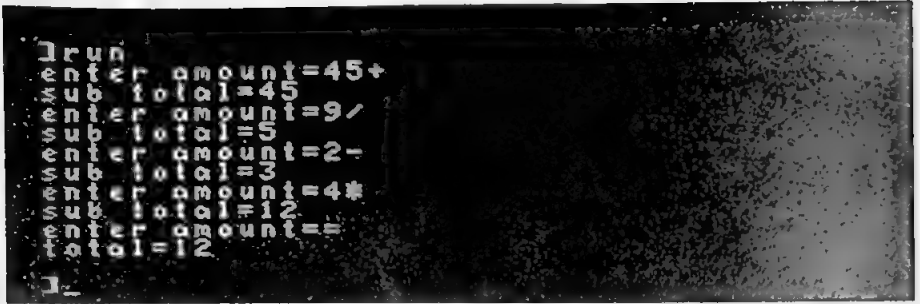
To help you distinguish between variable names and commands, smartBASIC displays all commands as capitals and all variable names in lowercase. For example;

PRINT amount

PRINT is a command name and *amount* is a variable name.

Type Variable	Description									
Decimal	Used to store decimal numbers. Examples: a=123.45 amount=-.0005 Large and small amounts are represented in the form .number E .exponent. The exponent represents the number of places to shift the decimal point to get the actual number. Example: <table><tr><th>Actual number</th><th>Exponent form</th><th>Comment</th></tr><tr><td>10000.</td><td>1E+4</td><td>Means add 4 zeros to the right of decimal point</td></tr><tr><td>.0001</td><td>1E-4</td><td>Means shift decimal point to left 4 positions</td></tr></table>	Actual number	Exponent form	Comment	10000.	1E+4	Means add 4 zeros to the right of decimal point	.0001	1E-4	Means shift decimal point to left 4 positions
Actual number	Exponent form	Comment								
10000.	1E+4	Means add 4 zeros to the right of decimal point								
.0001	1E-4	Means shift decimal point to left 4 positions								
Integer	Used to store whole numbers in the range -32767 to +32767. The percent sign (%) after the name identifies it as an integer variable. Example: a%=123 amount%=5 t1%=-32767									
String	Use to store characters from 0 to 255 in length. The dollar sign (\$) after the name identifies it as a string variable. Example: a\$="abc" char\$="z"									

FIGURE 5-7 Summary of Variable Types.



```
run
enter amount=45+
total=45
enter amount=9/
total=54
enter amount=2-
total=52
enter amount=4*
total=208
enter amount=
total=12
```

FIGURE 5-8 Output of Final Calculator Program.

SAVE the program and RUN it. Figure 5-8 shows a sample run that tests all the different arithmetic operators. Test your program to make sure you get the same results.

If you want the output to list on the daisy-wheel printer, type PR#1 before RUNning the program.

Now you know what BASIC programming is all about. If you're still confused, try rereading. Sit down with an ADAM computer and follow the instructions you have just read. There is no substitute for actually using a computer to test the commands and write the programs.

CHAPTER SIX

More Things You Can Do with SmartWriter

SmartWriter is so easy to use that you may think you don't need instructions or help. Unfortunately, many people take this approach to learning and end up never using an instrument like a word-processing computer to its fullest. In this chapter you'll learn how versatile SmartWriter is, and how many things you can use it for: writing, keeping simple records and, believe it or not, helping you write and edit BASIC programs. Some SmartWriter features will be more valuable to you than others, but you won't know that until you've examined them.

Every feature and use discussed in this chapter is illustrated with actual examples to help you master SmartWriter. However, you should practice with your own writing, to test your own experience, as you read the text.

GOOF-PROOF COMPUTING

Sometimes you'll do something you didn't really want to do. Don't worry, nothing breaks and almost every action has a remedy. It's important to remember that you've done nothing wrong. Learning to use a computer is like any other skill, whether skiing or learning to operate a microwave oven, and it takes a little time and patience. ADAM's goal is to serve you, so don't feel guilty, incompetent or stupid. You can do it, and it's fun!

There are three important keys to get you out of nearly any dilemma. The BACKSPACE key fixes nearby errors. Unless you're an

amazingly good typist, you're bound to hit the wrong key from time to time. The BACKSPACE is most useful for correcting errors either on the platen or on the two lines of text you're working on right now. Type the following lines, with the errors:

I think that I shall nevver see RET
A poem lovey as a tree

The cursor is now behind the last e in "tree"; press the up-arrow cursor key, and the first line will come back down onto the platen. Move the left-arrow cursor key under the second v in "nevver" and press BACKSPACE. Move the cursor back to the end of the second line with the cursor keys; if you used the space bar or began typing now, you'd type right over the letters that are already there.

The second key is UNDO, which reverses the last DELETE, CLEAR or BACKSPACE you did. Do you still have Joyce Kilmer's poetry on your screen? Press CLEAR and follow ADAM's instructions until it is gone. Now press UNDO and you have it all back! Just remember to press UNDO *immediately*, because if you press any other key first you cannot undo what you did.

The third key is ESCAPE/WP, which does not correct errors or restore deleted text, but instead takes you out of a function you don't want or chose by mistake. Press this key any time you want to stop a process you initiated with function keys I through VI (the "smart keys") or any of the six command keys at the top right corner of the keyboard.

Joyce Kilmer wrote his famous poem "Trees" during World War I, where he lost his life. Kilmer never had the opportunity to write his poetry with word processing, since it would be another 25 years before the computer was invented, but he certainly would have appreciated how easy it is to correct misspellings. Did you wonder if the word *lovey* was correct? You're right, it's supposed to be *lovely*. Move the cursor to the letter y and press INSERT. Type the letter l, then look at the screen. Smart key VI reads ALL DONE, and since you are, press it. You've made the correction!

Now let's move the cursor to the end of the line, but in a different way. Press the right-arrow key and the HOME key *at the same time* and you'll see the cursor scoot right to the last letter. You must press both keys simultaneously to reposition the cursor; pressing HOME alone takes you to the one of the first visible lines on the screen, and pressing the arrow key alone moves you a line or a character at a time. Type a period at the end of the sentence, then STORE the poem.

Press key V, STORE WORKSPACE, select drive A on key III, name your file "Trees," and press key VI for final STORE. The tape will spin, and in a few seconds your file will reside safely on the tape.

Moving and Copying Text

Here's an exercise to show you how you use the HI-LITE smart key to move text. Type the following:

Isaac Asimov, the famed science fiction author, published a book of short stories in 1950 called "I, Robot," that described the Three Laws of Robotics:
RET

1: A robot may not injure a human being or, through inaction, allow a human being to come to harm. RET

3: A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

2: A robot must obey the orders given it by human beings except where such orders would conflict with the First Law. RET

You probably noticed that Laws 2 and 3 are out of order, so let's fix that. Press MOVE/COPY; you'll see a message on the left asking you to HI-LITE the text you want to move. You must move the text back onto the platen with the up arrow until the cursor is on the number 3 for Law 3. Then press key IV and you'll hear two bells and see a red line appear under the 3. Move the cursor down to the period at the end of the sentence, under the RETURN arrow, and press key V. You'll hear two bells again. This underlines the entire sentence and concludes the HI-LITE process.

Now you'll see the MOVE message on the screen, so move the cursor below the last sentence (Law 2). Note the text has returned to the top of the First Law, so you have to move all the way down again. You can move one line at a time with the cursor, or you can press the down cursor key and HOME at the same time to move to the end of the text. When you're at the first blank line on the platen, press key VI, MOVE, and the Third Law will move into its proper place. This is what you should see:

Isaac Asimov, the famous science fiction author, published a book of short stories in 1950 called, "I, Robot," that described the Three Laws of Robotics:

1: A robot may not injure a human being or, through inaction, allow a human being to come to harm. RET

2: A robot must obey the orders given it by human beings except where such orders would conflict with the First Law. RET

3: A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Asimov's Three Laws of Robotics (which, by the way, have all been broken!) should look perfect now.

How to Delete

Let's try an exercise in deleting. Move the cursor to the beginning of Law 3 and press DELETE. Press key IV to HI-LITE and move the right-arrow cursor down the first line. Notice it comes to the end of the line and continues on to the second line, just like your words when you're writing. But it gets to the end of the second line and stops. Why? Because *you can only HI-LITE to delete what is in view on the platen*. Press the down-arrow cursor; the third line appears and you'll see the cursor, and the red underline, at the *end* of the sentence. Handily, you can HI-LITE in either direction, so press the left-arrow cursor and finish underlining. All through? Press key IV, HI-LITE OFF, then VI, FINAL DELETE.

ADAM has now deleted Law 3 and returned you to the beginning of the file. Let's get it back. Press UNDO and it will reappear exactly where it was. Just remember you must press UNDO *immediately* after your mistake, before you touch another key, or whatever you deleted is lost forever.

Now that you have all three laws in their proper order, STORE the file and, if you wish, PRINT it out. The safest way to protect your files is to store them right away, using STORE WORKSPACE, before you do anything else like printing.

WRITING LETTERS

You've now learned the basic word-processing skills: entering, deleting and inserting text, how to correct errors, saving text on the data cassette and getting it back, and printing. If you had *any* problems, keep practicing. ADAM is a computer, and computers are very

precise. You might have struck the wrong key, even though you'd swear you didn't! Or you might have had the cursor in the wrong place, or forgotten to underline all the text, or something else. Usually, because computers are so precise and humans aren't, the problem is human error. Remember the clerk in the department store who said, "It wasn't my fault, it was the computer!" Do you still believe her?

If you're like most people, you probably write a lot of letters—but probably fewer than you'd like because it's such a hassle. Now that you have ADAM, it's no longer a hassle; it's fun. Let's try a few to get the hang of it.

Do you have your own letterhead stationery? If not, let's have ADAM and SmartWriter design a handsome letterhead for us. Is your workspace clear? Good; now spell your name exactly as you want it printed on a piece of paper. Here's a sample:

Denman B. Engstrom
Ballymoor
266 Mansfield Street
Escondido, CA 92025

When you're all through writing your name and address, STORE it on the data cassette. Name the file *letterhead*. Now, whenever you want to write a letter, GET the *letterhead* file and it will appear on the screen. Write your letter and STORE it under a different name; ADAM stores the original *letterhead* file under its own name, ready for use the next time you write a letter!

Let's try writing a short business letter now. Business correspondence takes many forms, including complaints and compliments, inquiries, selling and confirming sales, but all good business letters consist of the following elements:

- The date
- Name and address of person to whom you're writing
- Salutation
- Paragraph 1: the occasion, or reason you're writing

- Paragraph 2: the facts you wish to express
- Paragraph 3: the action to take or that has been taken
- Paragraph 4: closing remarks
- The complimentary closing
- Your signature and typewritten name

Here is a sample thank-you or courtesy letter in what is called the *full block* style, which means every line begins on the left margin.

Denman B. Engstrom
Ballymoor
266 Mansfield Street
Escondido, CA 92025
October 6, 1984

>

>

Mr. John Brockman
The Home Video Center
12 West Main Street
Escondido, CA 92025

>

Dear Mr. Brockman:

>

I would like to thank you and your employee, Paul Moore, for helping me select and install a video cassette recorder in my home last Saturday. When I came into your store, I knew I wanted to get a VCR, but I didn't know what I wanted. In fact, I didn't have the slightest idea what I was getting myself into!

>

You asked me a few questions about how I planned to use the VCR and explained some of the features you thought I would find useful. That made me feel comfortable about buying the VCR.

>

The machine you helped me select was just right for my needs—and my wallet! And the real frosting on the cake was installing it. Paul Moore brought the VCR to my home and had it hooked up in five minutes. But best of all, he explained how it works so I could understand it. I've had it operate perfectly every time I use it, and that makes me very happy!

>

Again, thanks for your help with a very important purchase. I recommend

your store to any friend who plans to purchase video equipment. And by the way, I've joined the video tape club, too!

>
Sincerely,

>
>
>

Denman B. Engstrom

The other most common business-letter format is the modified block. Note that each paragraph is indented five spaces. This is done with the TAB, which you set up with key I, MARGIN/TAB/ETC. Press key IV, then move the right cursor key five spaces, press key III, and the tab is set. Look at the top of the screen for the <. You can set as many tabs as you like. Here's a slightly different letter in modified block.

Ballymoor
266 Mansfield Street
Escondido, CA 92025
October 6, 1984

Mr. John Brockman
The Home Video Center
12 West Main Street
Escondido, CA 92025

>
Dear Mr. Brockman:

I bought a video cassette recorder from you last Saturday and am writing to tell you about some problems I had. I hope that by sharing my experiences with you and your employee, Paul Moore, you can help other customers avoid the frustration I felt.

You asked me a few questions about how I planned to use the VCR and explained some of the features you thought I would find useful. That made me feel comfortable about buying the VCR.

Paul Moore brought the VCR to my home and had it hooked up in five minutes. He showed me how to turn it on and make it play or record tapes, but he didn't explain the timer and how it worked. I read the manual four or

five times, but I still couldn't understand it. Finally, I called him and he tried to explain it over the phone, but in the end he had to come to my home a second time so I could see how he did it.

You were both very helpful, but I don't think you want to have your employees making house calls over and over again. My advice is that you not assume people understand how to operate a VCR, even though it's a wonderful machine. If Paul had spent another 15 minutes the first time, I'm sure I could have figured it out.

I'm having no problems with the VCR now, and do want to say you and Paul Moore have been very gracious and helpful. I hope my experiences are helpful for your future customers. And by the way, I've joined the video tape club, too!

>

Sincerely,

>

>

>

Denman B. Engstrom

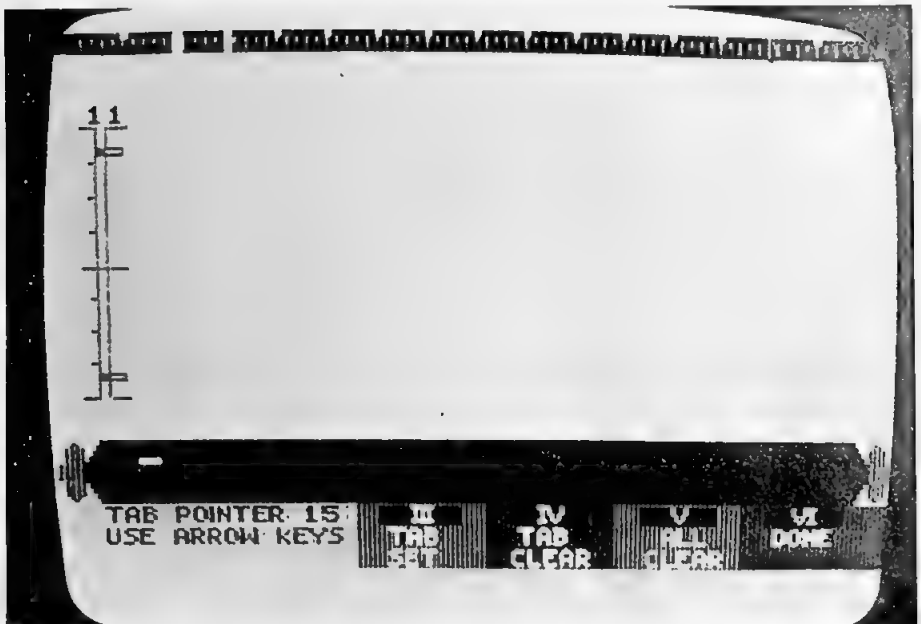


FIGURE 6-1 Tab Option Screen.

Search and Replace

How many times have you written something to find that you misspelled a word that you used over and over? Well, correcting errors with ADAM's SEARCH function is not only fast, it's fun! Type the following story, taken from a mother's diary about her children, and you'll see how SEARCH works.

Joshua's first day at nursery school was eventual, to say the least! I had bought him new clothes, a pair of Jordache jeans and a bright-colored rugby shirt, so that he would make a good impression on the nursery schoolteacher, Miss McMorrow. When we arrived, Joshua went straight for the sandpile. That would have been all right any other day, but it had rained the night before and the sand was dirty and wet. Before either Miss McMorrow or I could stop him, he was a mess.

We brushed him off as best we could, then went into the nursery school playroom to meet the other children. One darling little girl was playing with fingerpaints, and I don't have to tell you what happened to Joshua next. When Miss McMorrow served cookies and milk, Joshua must have thought his shirt was a napkin, for it was covered with stains.

I probably should have dropped him off at the front door and just gone home, but it was his first day and I wanted to take pictures of my sweet little boy at his nursery school. Miss McMorrow must have sensed how upset I was, for she told me most of the children made a mess of themselves on the first day for some reason she couldn't guess. She smiled brightly and I cheered up and took some pictures anyway. Mess or no mess, he's still our Joshua!

You probably noticed that *nursury* is the misspelled word. Move the cursor, using the HOME and up-arrow cursor key, to the beginning of your document. Press key III, SEARCH; the message box asks what word you want to search for, so type the misspelled word. Press key VI to BEGIN SEARCH. Notice how quickly ADAM searches for the word? The message screen reads SEARCH COMPLETE, and gives you three options. Select key V, REPLACE, and type the correct word, *nursery*, on the platen. Since you know the word is misspelled several times, you might want to select key VI, REPLACE ALL. Now watch what happens! ADAM reads the document and every time the incorrect word shows up, it's spelled correctly, automatically.

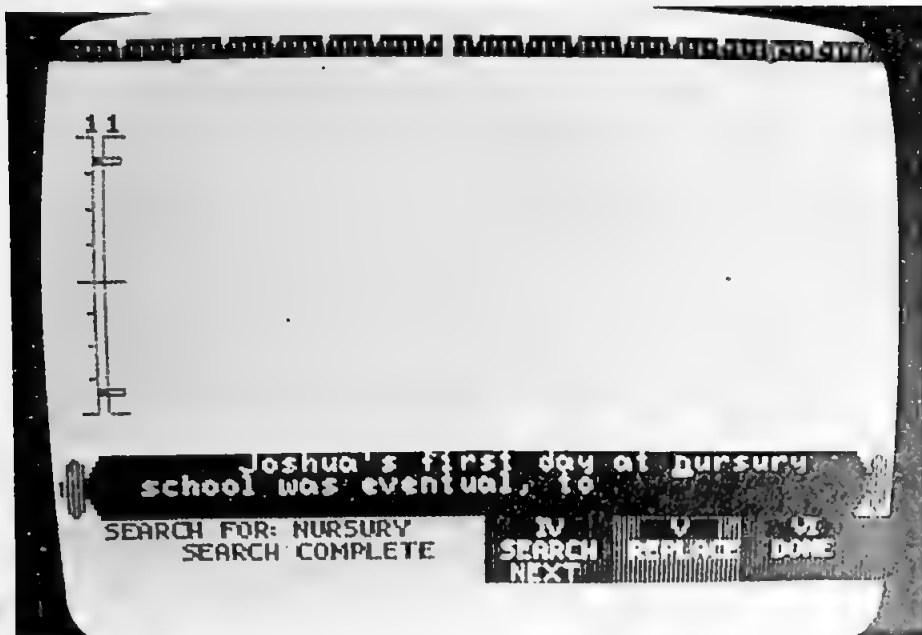


FIGURE 6-2 Search Option Screen.

If that was so much fun you'd like to try it again, go back to the beginning and choose another word to search and replace, like Joshua or the teacher's name.

You can type the word you're searching for either with or without capital letters, as you wish; ADAM will find the correct word whether it's capitalized or not. However, you must type the word with which you plan to replace the incorrect one correctly, just as you want it to appear. For example, you could SEARCH *mcmorrow*, but you should REPLACE *MacMorrow*.

SEARCH is useful in many ways. If you're writing a long term paper you can use SEARCH to find ideas, terms, names, facts, anything you like, to check or reference them. Say you're writing and come to a place where you're not certain about something, but you have to go to the library to get the information. Put a special mark such as] or > or + in the text—something you normally wouldn't type. When you return from the library, SEARCH for your mark and type in the missing information. SEARCH is a quick way to read through a long term paper or report or the book you're writing; all you have to do is search for key words or footnote numbers.

WORKING WITH LONGER DOCUMENTS

ADAM and SmartWriter are hard workers. Together they will do just about anything you ask of them, so long as you're careful and logical. Assuming you will respect these two rules, let's write an essay that runs several pages long and learn some more things about SmartWriter. Remember to begin by typing a RETURN so ADAM will be sure to print the first line.

Learning About Word Processing

By
Lori Tanzer

When historians talk about the world's most significant inventions, one machine you don't often hear mentioned is the typewriter. Who would praise a machine that nearly everyone hates? Christopher Latham Sholes, an American, developed the first useful commercial typewriter in 1868. It employed what is called the QWERTY keyboard, designed to slow the typist down so that the key bars would not collide with one another. This keyboard is still with us today, even though there aren't many typewriters being manufactured with regular key bars anymore.

It is interesting to read the first letter ever written on a typewriter, 38 years before Sholes perfected his commercial machine. A man named W. A. Burt had invented something he called a typographer and wrote this letter to his wife on March 13, 1830:

Dear Companion, I have but jst got my second machine into operation and this is the first specimen I send you except a few lines I printed to regulate the machine.¹

Let's stop for a moment to see how the subscript/superscript function, key VI, works. Press key VI and you'll see two options. SUBSCRIPT types words or letters half a space below the regular line. This is normally used for chemical or physical properties, such as H₂O. SUPERScript types characters half a space above the line and is used most often for references in text, powers of ten (4²) and trademarks (ADAMTM). We're going to number our references, or footnotes, so type VI for SUPERScript. ADAM now asks what text

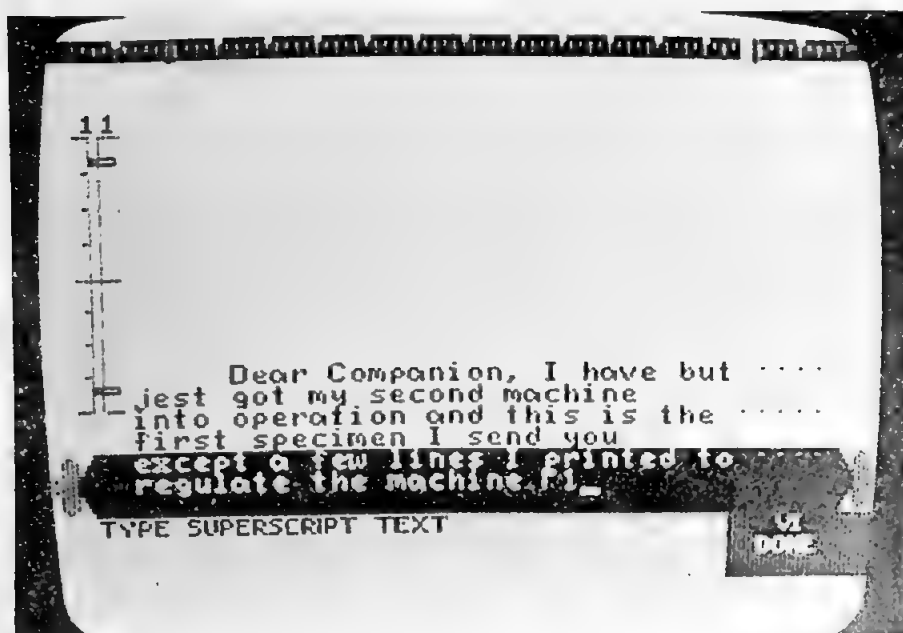


FIGURE 6-3 Superscript/Subscript Option Screen.

you want to type, so type the number 1. Press V1 again for DONE and you'll see the 1 in brackets on the screen. When you print the essay, the 1 will appear just above the line! Watch for the next three references and put them in superscript as well.

When you're working with a long document, it's a wise idea to STORE every so often. The *Companion* suggests you pause to STORE after you've typed about 50 lines, or at the end of every third screen. This protects your writing from almost any calamity—a power failure, acts of God, or the baby accidentally pressing the DELETE key.

STORE what you've written thus far in the normal fashion, and name the file *essay*. Once ADAM is through storing, you can continue to write on the screen, right where you left off.

His words are a far cry from what Thomas Watson said to Alexander Graham Bell the first time they got the telephone to work: "What hath God wrought!"²

The concept of word processing dates back to 1964 at IBM.³ The idea was to eliminate repetitive typing tasks where, for example, one letter had to be

sent to a dozen people or where standard terminology, often referred to as "boilerplate," was used over and over again. Law offices use boilerplate for contracts and other legal documents all the time, and were one of the first to begin using word processing.

Early word processors consisted of a Selectric typewriter hooked up to a magnetic tape-recording machine that "recorded" letters and other written material and then "played" it back. The typist had to press certain keys in a complicated order such as CONTROL S/ then SHIFT 23 to command the word processor to enter text, retrieve files or print. It took many hours to learn these commands—sometimes there were hundreds—and many simply gave up and typed the letters one at a time, over and over.

Today's word processors are much easier to use. On most you simply type as you would on any typewriter, but when the words reach the end of the line you don't have to hit the carriage return. The words continue to spill onto the next line as you watch the monitor screen. If you want to correct a word or change a phrase, you simply move the cursor—that blinking light on the screen—to the right place and type over the offending words. You can insert, revise or delete your prose until it's just how you want it by pressing keys marked INSERT, MOVE and DELETE. Then and only then do you print it on paper. No more wastebaskets full of paper balls.

The writer H. L. Mencken once said, "There is in writing the constant joy of sudden discovery."¹ After 100 years of toiling at the typewriter, we now have an alternative that is fun to use and that frees us to be more creative.

Now STORE again to protect your work. You can continue writing and editing the text on your screen and periodically re-STORE it, which is easier than clearing and getting over and over. Remember, you must STORE and name the file (*essay*) the first time, then CLEAR the screen and GET *essay* to re-STORE and continue writing it; don't change the file name each time.

Now we'll type a final, separate page for the footnotes. Go to the end of the essay and press key I, MARGINS. Select key VI, END PAGE, and that's it! You'll see an E appear in the left margin, signifying the end of that page and the beginning of another. Now type the footnotes page. To underline the book titles, type the words, then move the cursor back to the beginning of the title, then hold the SHIFT key down and press the number 6. It works just like a typewriter.

References

- ¹Maureen R. Jacoby, *The Official Guide to the Smithsonian* (New York: CBS Publications, 1976), p. 29.
- ²Roger T. Houghlum, *Electronics: Concepts, Applications and History* (Boston: Breton Publishers, 1980), p. 14.
- ³Robert Sobel, *IBM: Colossus in Transition* (New York: Times Books, 1981), p. 304.
- ⁴John Bartlett, *Bartlett's Familiar Quotations* (Boston: Little, Brown, 1980), p. 771.
-

All through? Then STORE one final time; this is your finished document. CLEAR the screen and GET *essay*.

Our essay followed standard rules and conventions for footnoting and referencing, but check with your teacher before you write anything for your class to make sure you follow his or her rules. We're going to PRINT our essay double-spaced, which is the usual practice, but some teachers won't want you to double-space the reference. If so, make the reference page a separate file and print it out by itself.

Before printing, you should edit and proofread the essay, to correct spelling errors and change any words or phrases. Normally, you would have to correct a line at a time on the platen, but to make this task a little easier and a lot faster, let's edit in the moving window format.

When you're writing original text on the platen, your screen shows only 36 columns or characters. The moving window shows an entire page, 80 columns or characters. It's just that some of the page is displayed off the screen. Select key II, SCREEN OPTIONS, then key VI, MOVING WINDOW. The entire screen will fill with text and you can scan through it by moving the cursor. Now you'll appreciate the cursor arrow/HOME key combination even more. Press the right-arrow/HOME keys *simultaneously* once; you move to the right side of the text. Press them again and you move to the rest of the line, the right half of the page. Press the down-arrow/HOME keys and you

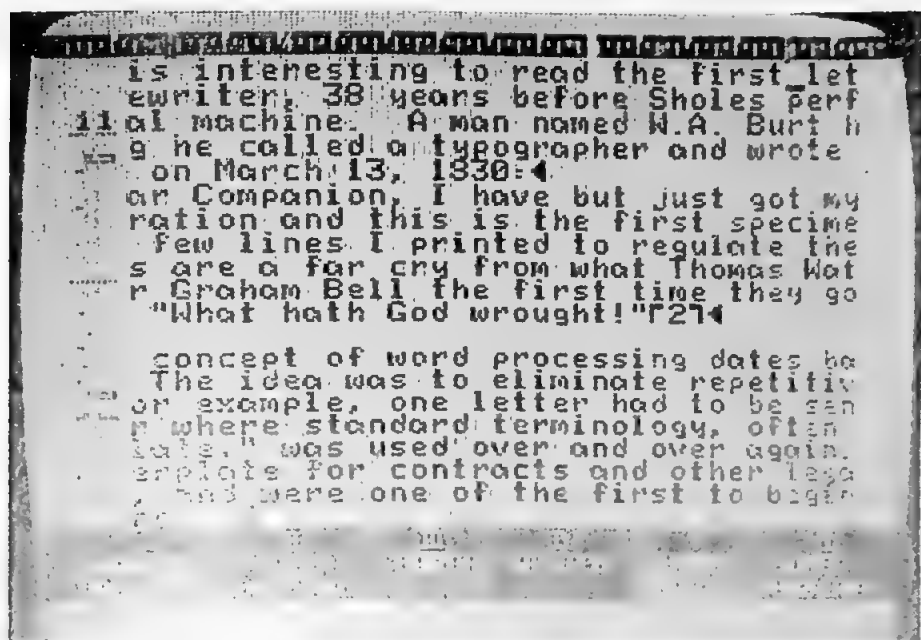


FIGURE 6-4 Moving Window.

move first to the bottom of the screen, then to the next frame of 20 lines. Of course, you'll want to move the cursor through the text without the HOME key to make corrections.

Make all the changes you wish, then STORE the file one last time. You can STORE in either MOVING WINDOW or STANDARD format, it doesn't matter. Once the essay is safely stored, GET it back for final printing. When it's on the screen (actually, you'll only see the first line, which is the RETURN you put in), press key I, MARGIN, and select key V, LINE SPACING. ADAM can print single, double, or triple space and half spaces in between. Press key V twice to select double space, then key VI, DONE.

Now press Print, and key V, PRINT WORKSPACE. You'll see the message window reads SINGLE SHEET. This is the first time we've printed anything longer than one page, so we have a choice. If you're using regular paper, leave everything as it is. If you're using continuous-form paper, press key II, FAN FOLD, and ADAM will continue printing each page automatically. Choose your option, then press key V to begin printing.

WORKING WITH MORE COMPLEX FORMATS

Everything you write won't be as simple and straightforward as the essay above: term papers with long quotations, poetry, resumes and documents with lists require special margins, or *formats*, as we term them. From time to time you may want to insert the same paragraph or information again and again, which is called *boilerplate*. ADAM can do these things for you if you're careful; remember, your ADAM computer is not as sophisticated as the computers in offices, which usually cost ten times as much.

ADAM can format only one set of margins in a document. That means you can't change the left or right margins within a document; if you do, you change the margins for the entire document.

This is a bit cumbersome when you're writing a term paper and must insert text from another source, which you're required to offset from your essay. However, you can easily get around this with the TAB. By now you're probably comfortable writing in *standard format*—that is, on the platen. When you begin manipulating different blocks of text, it's much easier to shift to the *moving window*. You can keep an eye on the cursor and note precisely where it is in the text, which, you'll soon see, comes in handy. Press key II and select VI, MOVING WINDOW.

Begin your writing and continue until you've reached the spot where you wish to place indented text. Now press key I, select TAB and indent five spaces (or more if you wish). You'll see a red marker appear on the line at the top of the screen. Type in the first line, watching the cursor move across the top of the screen. When you see you've reached the white right-margin mark, press RETURN. The cursor drops to the line below; now press TAB and continue typing until you reach the right margin mark again. Here's an example.

Mary Driscoll: A Modern Lyrical Poet

Mary Driscoll's poetry is evocative, elegant and deeply Irish. She blends William Butler Yeats's lyricism, Dylan Thomas's cynicism and the stream-of-consciousness archetypes found in James Joyce into rare excursions in contemporary poetry. A resident of Millbury, Massachusetts, Driscoll gradu-

ated magna cum laude from Smith College, then took her master's degree at Wellesley College. She has written poetry all her life while teaching, running the family business and raising ten children, one of whom was lost in a tragic accident. Her poems have appeared in a number of literary journals and in two published volumes, *In Formal Gardens* (Rowe & Co.) and *Brief Lightning* (Dorrance & Co.). God's ways, the inexorable aspect of human nature and the pastoral view of life all play a part in Driscoll's poetry. For example:

QUAN YIN

I was a mandarin lady
when the thousand-year-old bird's egg
lay hidden in a nest in an almond tree.

I poured jasmine tea
for some of Marco Polo's friends,
and together we walked in the garden
when the dawn mist silvered the plum.

My feet were never bound
nor my mind.
I knew the final sampan's destination
and I watched the Bible people come.

Now I'm inscribed in an ivory shawl,
struck graceful, changeless as porcelain.
Glazed like Heidegger by layers of life,
I'm stuck in forever.

But sometimes the muted beak
pecks shyly at the shell,
remembering how it was
to shimmer and be green.

Note how religion and philosophy mix and intermingle in the poem: the nearly religious Oriental tea ceremony and the Christian missionaries clash with Heidegger's atheistic existentialism. Nature images abound in the bird's egg, the almond tree, the plum, the ivory.

The poem was a fairly simple insertion, since none of the lines ran too long and each began without further indentation. But you can do more complex insertions with ADAM just as easily; just be sure to watch your spaces.

In another of Driscoll's poems, less artfully abstract but mystical nonetheless, we see a closer glimpse of the poet's feelings:

AFTER FAMILY FUNERALS WHEN THE YOUNG DIED

My catafalque's a bobsled,
red
 and ash
 and shiny,
curved neatly,
 no slivers
 no heavy timbers,
 no crossbeams.

'Round and up and down,
I and my catafalque
 slide
 and roar
 and sigh,

and we almost always
 make it safely
 into the wind tunnel

when the avalanche shudders by.

In this case the poet, following a literary convention, added two spaces to each succeeding line, so all you must do is hit the space bar to advance the cursor to the correct place each time. ADAM prints the text out just as you see it in the moving window.

Correcting Errors and Editing in Moving Window

It's a lot faster and easier to edit a long essay or document using the moving window because you have all the screen filled with text, as opposed to just the two lines visible on the platen in standard format. The only disadvantage, however, is that you see only half the page in the moving window. That's easy to adjust to once you begin working with a document.

If you press the right arrow key, you'll see the cursor zip to the right edge of the screen and then jump to the text that wasn't visible on the screen. You can scan each line in this fashion, but you might prefer to press the HOME and right-arrow key together. The first time you press them the cursor jumps to the right side of the screen; press them again and the entire block of text to the right appears. It's less jumpy than the scanning technique, and to return to the block of text to the left you simply press HOME and the left-arrow key.

When you find errors, use the BACKSPACE, DELETE or INSERT keys to make your corrections. You can insert or delete spaces in the same way you do letters or numbers. You'll find the search-and-replace function works especially well in the moving window. Say you misspelled a word that appears several times in your document; search and replace finds it every time it appears and spells it correctly!

OTHER USES FOR SMARTWRITER

If you're like most people, there are probably more things you'd like to do in life than you have time for. SmartWriter is an information-management tool that can help you record and store valuable facts and data. You might assign your family's health records to one data cassette and keep a separate file for each family member. Each time someone goes to the doctor, has an illness or an inoculation, or anything else of note occurs, you can write it in the file in chronological order. If you must review someone's records, you can search by date or by illness and quickly locate the information you need.

You can store household inventory information for insurance purposes in the same manner: whenever you buy something, add it to the list; send a copy of the list to your insurance agent each year with the renewal to help determine if you need more coverage.

You're probably thinking of other things you want to catalog already; we all have collections of one kind or another, and often don't write down all the ideas or supplemental information we'd like to. Now it's quick and easy with ADAM and SmartWriter.

Once you become proficient with SmartWriter, you may even want to turn your interests into a small business. After all, others need information, which you could gather and provide for them on a regular basis. The local video store might want to keep track of how frequently certain films are rented, or the druggist might want to see

how successful one special is versus another. You can keep a running tally from the receipts or coupons they provide.

Many proficient typists run a transcribing service for doctors, attorneys and businesspeople, typing notes, letters and important papers from cassette tape recordings. Think how efficiently you could do this with ADAM and SmartWriter! You might strike up a working relationship with the owner of your local copy shop to prepare custom-formatted or personally addressed business letters. Using the program the *Companion* provides in Chapter 9, you can create or compile mailing lists for businesses in the same way you do for your Christmas cards.

You might open a resume-writing service, creating a perfectly printed original which the copy shop reproduces for your customer. Here is a sample resume:

Garrow Throop
244 Ridge Road
Santa Fe, New Mexico

Home (505)633-4595

Work (505)629-1800

EMPLOYMENT OBJECTIVE: Magazine Design and Layout

EXPERIENCE

1981-present Arthur Samuels Corporation, Santa Fe, NM
Staff Designer. Design company stationery, new product packaging, logos, etc., for educational software firm. Responsible for design and layout of monthly newsletter distributed to dealers and customers.

1978-81 *Santa Fe Magazine*, Santa Fe, NM
Senior Designer. Responsible for all aspects of magazine layout, design, and production. Began as graphic artist and was promoted twice. Magazine ceased publication in 1981.

EDUCATION

1972-76 University of California at Santa Cruz
BFA, Fine Arts and Graphic Design

1976-78 California Institute for the Arts and Sciences
MFA, Fine Arts and Graphic Design

AWARDS

1983	<i>InfoWorld</i> Magazine Best Software Package Design
1983	Software Institute Best Software Package Design
1982	Newsletter Association of America, Best Institutional Newsletter Design
1978-81	Six <i>Folio</i> awards for cover design, lead art design, and best overall magazine design

INTERESTS Oil painting, photography, music, MTV

REFERENCES Available upon request

You may think about starting a newsletter for the PTA, your ADAM user group, or the civic, charitable or religious organization you belong to. With SmartWriter, you can create the narrow columns that allow you to become, in effect, your own typesetter; simply print them out, cut and paste them on 8½ × 11 sheets of paper, and have the copy shop duplicate them for distribution. Use your mailing-list program to print out labels.

You may even want to write and publish your own book. A group of women in Louisiana self-published a collection of their recipes, which has been reprinted 32 times and has sold over 200,000 copies. You and your friends may want to self-publish your own book of programs you've written for ADAM.

USING SMARTBASIC AND SMARTWRITER TOGETHER

ADAM has a wonderful feature: you can edit your BASIC programs in SmartWriter! While it's not hard to correct and edit lines of code in SmartBASIC, it's even easier in SmartWriter, since you can write over the old lines just as if you were using word processing. Write your programs in BASIC as you normally would, but if you're merging two separate programs (something you'll do often if you work with programs in the *Companion*), use SmartWriter's editing functions. Or if you wish to correct or change a word that appears repeatedly, use the search-and-replace function.

Here's an example of how you'd combine two BASIC programs

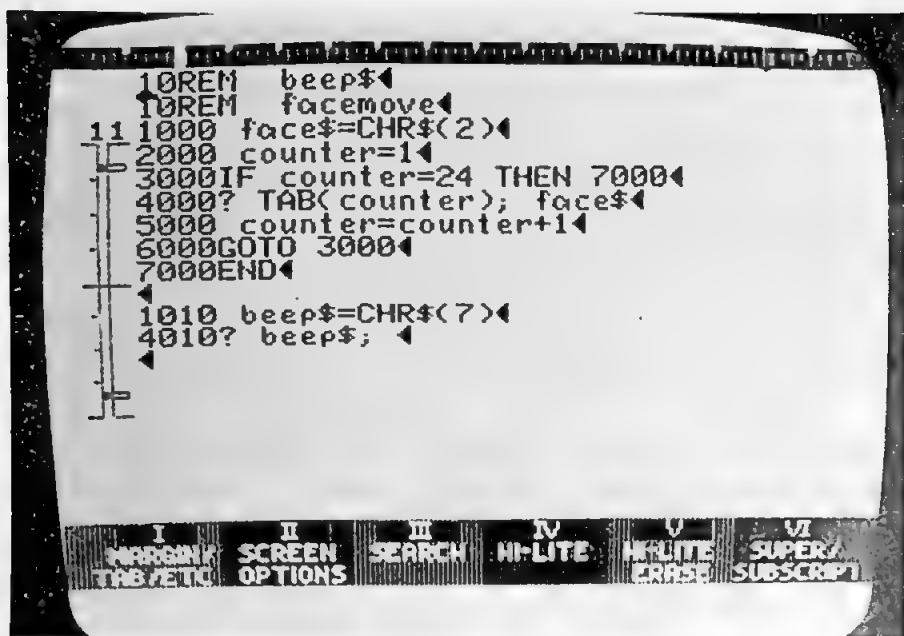


FIGURE 6-5 Merged Programs.

using SmartWriter. First, load your SmartBASIC tape and write this program:

```

10 REM beep$
1010 beep$=CHRS(7)
4010? beep$;

```

SAVE it and name it *beep\$*. Now write the following program:

```

10 REM facemove
1000 face$=CHRS(2)
2000 counter=1
3000 IF counter=24 THEN 7000
4000 ? TAB(counter); face$
5000 counter=counter+1
6000 GOTO 3000
7000 END

```

SAVE it and name it *facemove*. If you're using two tape drives, this

procedure is much easier since you can leave SmartBASIC in drive A and your program tape in drive B.

Now open the tape drive door(s) and pull the reset switch toward you to reboot ADAM. Press ESCAPE/WP to enter SmartWriter and close the tape drive door(s). Press STORE/GET, then key VI for GET. Bring your *beepS* file up on the screen, then GET your second file, *facemove*.

Do you have both files on your screen? Good!

Now STORE the combined programs under file name *facebeep*. Pull the reset switch and reboot SmartBASIC. Once it's loaded, type *catalog* (and D2 if you have two drives), then LOAD *facebeep*. Type LIST to see if all the lines are there. They are? OK, then type SAVE *facebeep* and watch what happens. The lines rearranged themselves into proper order, right? Now type RUN and watch your program work.

That's it. If you wanted to change the word *face* to, *say*, *button*, you'd go back to SmartWriter, then press key III, SEARCH, and follow instructions for search and replace. You'll find editing and correcting long programs much easier and faster in SmartWriter, and you'll probably learn a few tricks you could teach the *Companion* as well!

How to Create Your Own Picture Maker

We have all enjoyed drawing colored pictures as children. With ADAM you become an artist with your computer. Instead of using crayons or pastel chalk, you use the cursor control keys as your paintbrush and the function keys to select a color, store and get your picture. You can combine your pictures into a slide show to entertain, teach or advertise.

While ADAM has SmartWriter to process words, it does not come with a picture processor program. If you really want to create some beautiful high-resolution pictures, the *Companion* recommends Coleco's SmartPicture Processor or SmartLOGO, a powerful graphics command language that teaches children how to program. LOGO takes full advantage of ADAM's graphic and sound capabilities, which SmartBASIC does not.

Neither SmartPicture nor SmartLOGO shows you how ADAM creates graphics as does SmartBASIC. If you would like to understand computer graphics and draw some simple pictures using small colored squares, the *Companion* will teach you. You'll learn to write a picture-maker/editor program in BASIC and use BASIC commands to create and read a file—your picture—from tape cassette.

THE CANVAS

Your canvas is your color television or monitor screen. Figure 7-1 illustrates the screen layout. In the low-resolution graphics, SmartBASIC provides a grid 40 columns across and 40 rows down. In each

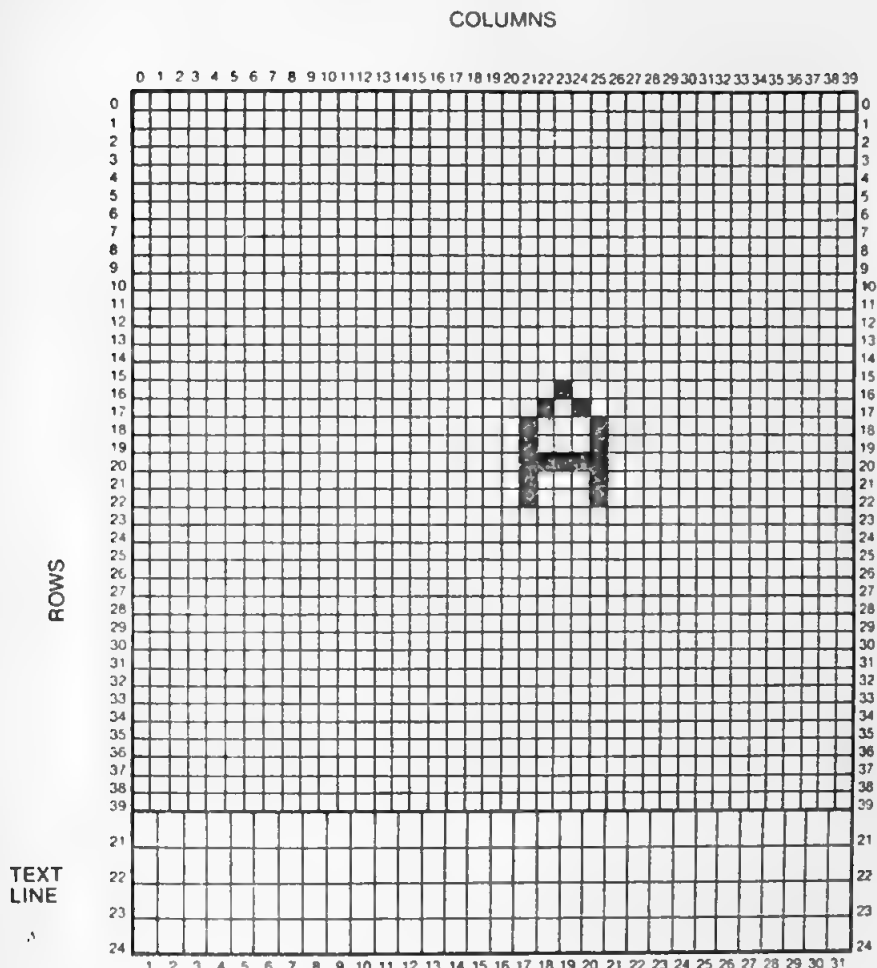


FIGURE 7-1 Low-Resolution Screen Layout.

box you can paint a dot in any one of 16 colors. The grid does not take up the full screen. Four text lines on the bottom of the screen allow you to enter commands or have your program's PRINT statements display instructions and error messages.

DRAWING THE BIG A

In Figure 7-1 you'll see a big letter A. In a moment, we'll draw it on

your screen. Appendix F has a blank layout form that you can copy to design pictures before drawing them on ADAM.

Load SmartBASIC and when it's ready you'll see the TEXT mode with 24 rows 31 characters wide. In this mode, you use the full screen to display text, letters and numbers. You can move the cursor to any of the 24 lines and enter a command or display text. To change to the low-resolution mode,

You type GR

The screen goes black and the] appears on the left, three lines from the bottom, text line 21.

To get back into text mode,

You type TEXT

Now the] prompt appears at the top of the screen.

To get back into the low-resolution mode,

You type — (fill in)

The first two letters of the word *G*Raphics.

In low resolution ADAM displays 16 different colors. After you draw the big letter A, the *Companion* will show you a program that displays the complete 16-color rainbow. To establish the color of the squares and lines you draw on the screen, you need to assign a color number to a special variable named COLOR. You can't PRINT or use it in arithmetic expressions but you can assign a value from 0 to 15 to it, defining one of the 16 colors shown in Figure 7-7. GR sets the COLOR=0. Zero defines the color black, but you can't see a black square on black background. To see the squares you're going to draw, you must set the COLOR to other than black. To set the color to dark green

You type COLOR=4

Nothing special happens, but now you can plot a square on the screen and see it as green. Plot the top of the big A at column 23, row 16.

You type PLOT 23,16

Now make the top of the A.

You type PLOT 22,17

You type PLOT 24,17

To plot a square you need to give a value for the _____ and then the _____. PLOT *column, row* allows you to put a color square in the box addressed by the column and row.

Now draw two vertical lines from row 18 through row 22 in columns 21 and 25, the left and right side of the big letter A.

You type `VLIN 18,22 AT 21`

You type `VLIN 18,22 AT 25`

To draw a thick colored vertical line you need to define the values for the beginning and ending _____ and the _____ in which to draw the line. `VLIN` from *row*, to row *AT column* draws a vertical line connecting squares in two rows.

To complete the A draw the horizontal middle bar connecting the two sides, and you have the big letter A. Draw a horizontal line from column 22 to column 24 at row 21.

You type `HLIN 22,24 AT 20`

You draw a horizontal by defining the values for the beginning and ending _____ and the _____ in which to draw the line. `HLIN` from *column*, to column *AT row* draws a horizontal line connecting two columns.

After all that work, you should see a large green block-letter A in the middle of your video screen as shown in Figure 7-1.

GOTO LOOPS

You remember that one of the three fundamental ways to combine processes is to repeat them. BASIC statements perform the processes, like adding numbers. In the pocket calculator program you used a `GOTO` command to repeat the input and arithmetic statements until the operator entered the equal sign. Repeating statements this way is called a *loop*. In the picture-maker program and other programs, you often need to repeat a group of statements to perform a process.

Figure 7-2 (next page) illustrates a more specific form of the loop that occurs when you want to repeat a group of statements a given number of times. Suppose you want to draw 20 horizontal lines every other row so you can verify that the low-resolution screen has 40 rows.

To clear any previous program in BASIC memory,

You type `NEW`

Let's call the program `LOOPING` and enter a `REMARKS` command to identify it.

You type `10 REM looping`

Use the `GR` command to set the screen to low-resolution graphics and assign the `COLOR` to dark green.

You type `1000 GR`

You type `1100 COLOR=4 : REM dark green`

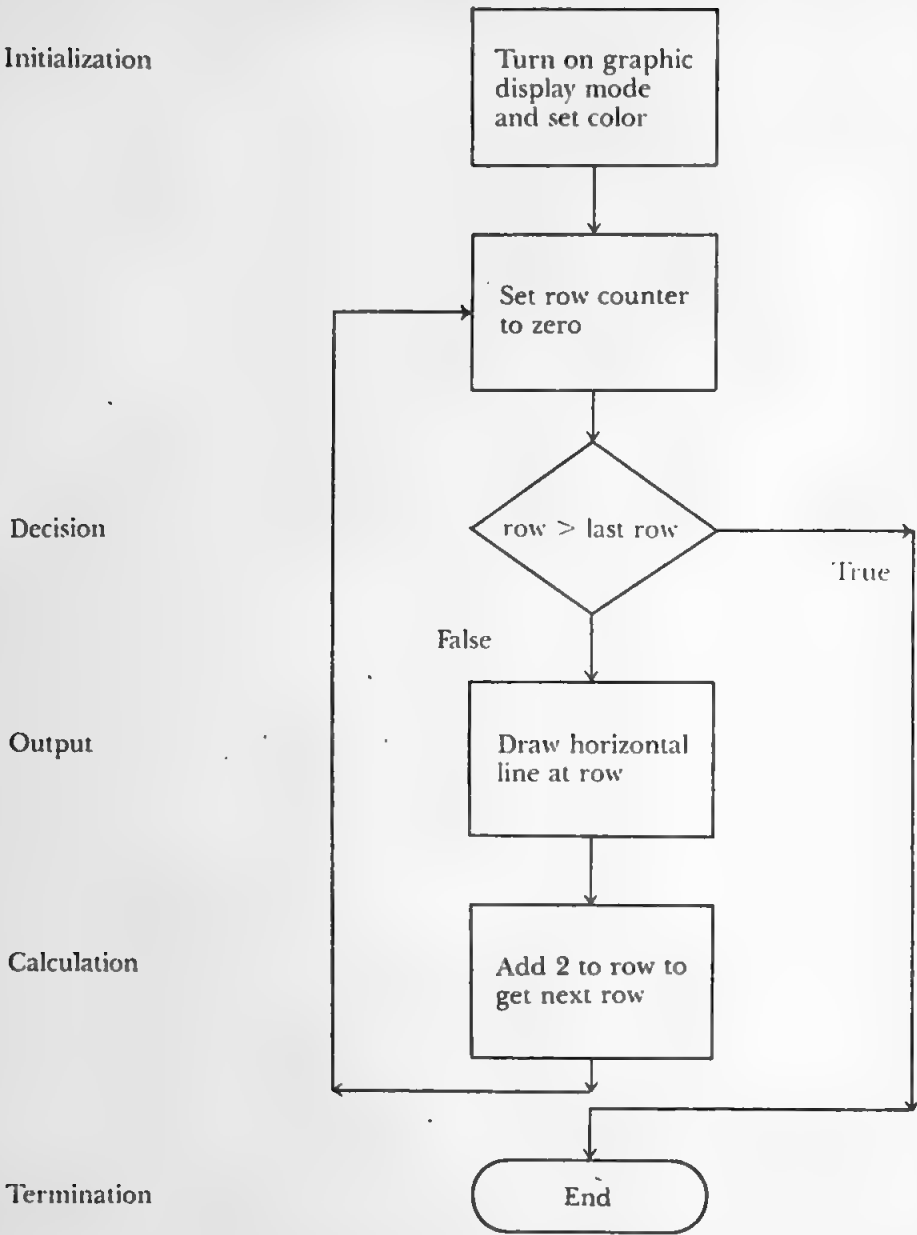


FIGURE 7-2 Flowchart of LOOPING Program.

You need an integer or decimal variable to act as a counter. We'll use a decimal variable named *row*.

You type 1150 row=0 .

The program GOTOs the termination routine when the counter, ROW, is greater than 39, the maximum low-resolution row. You use the IF command to make the greater-than test.

You type 1200 IF row > 39 THEN 2000

Use the HLIN command to draw the line from the left side to the right side in the current row.

You type 1300 hlin 0,39 AT row

Add 2 to the row, skipping odd rows.

You type 1400 row=row+2

Then write the GOTO command to loop back to the IF statement to see if ROW is greater than 39, the last row.

You type 1500 GOTO 1200

The termination routine, executed when the IF condition test becomes true, consists of the END command which causes BASIC to return to the immediate mode.

You type 2000 END

Type LIST and check that it matches Figure 7-3 (next page). Make any necessary corrections and then RUN the program.

You will see 20 rows of green parallel lines on the screen as shown in Figure 7-4 (next page).

To get back to the full-screen text mode,

You type TEXT

FOR . . . NEXT LOOPS

To repeat a loop a specific number of times you need four operations:

1. Initializing the loop counter. You did this when you set the decimal variable *row*=0

2. Testing if the maximum value has been exceeded, indicating the program should exit the loop. You did this when you checked if the value of *row* was greater than 39.

3. Adding a step value to the counter variable. You stepped up the loop counter *row* by two so you drew the line only on even-numbered rows.

4. Looping back and repeating the statements until the loop is done. The GOTO statement did this and marked the end of the loop.


```
10 REM    looping
1000 GR
1100 COLOR, = 4: REM  dark green.
1150 row = 0
1200 IF row > 39 THEN 2000
1300 HLIN 0, 39 AT row
1400 row = row+2
1500 GOTO 1200
2000 END
```

FIGURE 7-3 LOOPING Program.

Repeating a group of statements a specific number of times occurs so often in BASIC programs that it provides an easy way to define the *loop* value and execute it.

Try the FOR...NEXT in the immediate mode to print 22 numbered lines. The colon (:) allows you to write more than one command on a statement line.

You type TEXT

You type FOR line=1 TO 22:PRINT line:NEXT

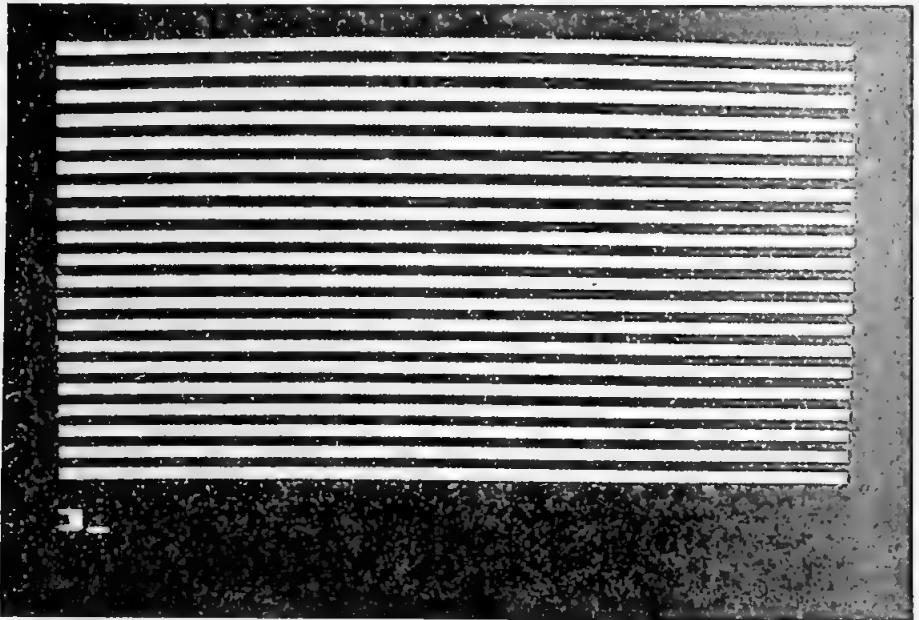


FIGURE 7-4 Looping Output.

Figure 7-5A shows ADAM displaying 22 number lines, space line and the] prompt.

You type ?line

ADAM displays 23

The NEXT command added the step value—BASIC used 1 by default—to the loop-counter *line* and repeated the loop until *line* was greater than the maximum limit counter value 22. When you printed the value of *line* after the FOR . . . NEXT loop, it had a value 23, which is greater than 22, and caused the loop to end.

You can have the loop counter step down from 22 to 1.

You type FOR line=22 TO 1 step -1: ?line: NEXT

ADAM displays the lines number 22 to 1.

You type ?line

ADAM displays 0

Figure 7-5B (next page) shows how the NEXT command added the negative step value -1 to the loop-counter *line* and repeated the loop until *line* was less than the minimum counter value 1. By displaying *line*, you see why the loop ended: *line* had the value zero.

When the FOR command has a positive STEP value, the NEXT command makes a _____ comparison with the _____ limit

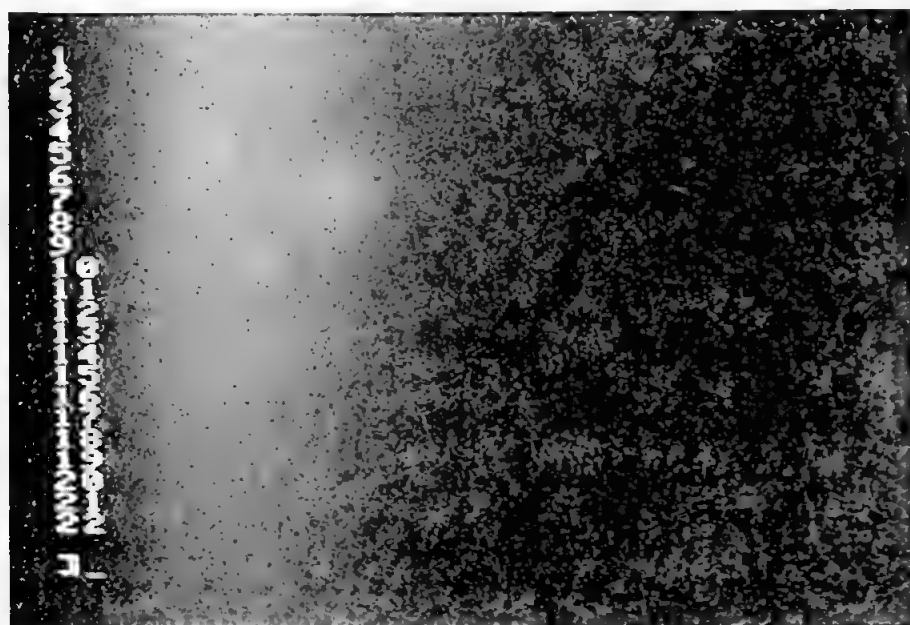


FIGURE 7-5A Positive For . . . Next.

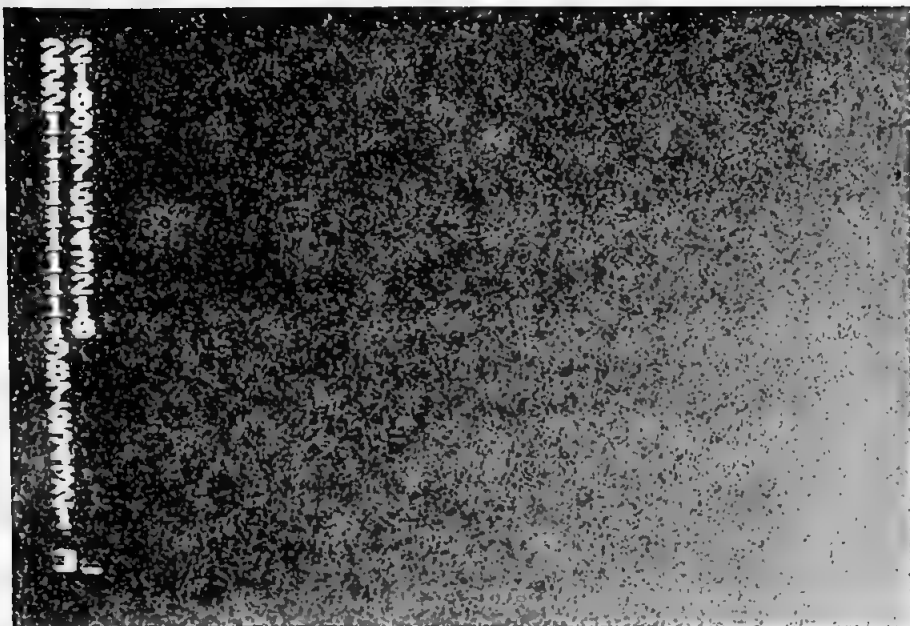


FIGURE 7-5B Negative For . . . Next.

counter value. It makes a *greater-than* comparison with *maximum* limit counter value.

When the FOR command has a negative STEP value, the NEXT command makes a _____ comparison with the _____ limit counter value. It makes a *less-than* comparison with the *minimum* limit counter value.

In a program you use the FOR . . . NEXT commands to define the beginning and ending points of your loop. You will see this in the RAINBOW program.

THE LOW-RESOLUTION COLOR RAINBOW

ADAM has a rich variety of colors, allowing you to create some very eye-catching pictures. The following program draws a rainbow of colors on your video screen and below each color lists the number you would use in the COLOR statement to get the color displayed. The *Companion's* programs use the words HUE or SHADE as the variable name used to store the color value.


```

10 REM    lowcolor
1000 GR
1100 hue = 0
1200 FOR column = 1 TO 32 STEP 2
1300 COLOR = hue
1400 VLIN 0, 39 AT column
1500 VLIN 0, 39 AT column+1
1600 hue = hue+1
1700 NEXT column
1800 HOME
1900 PRINT " 0  2  4  6  8  10 12 14"
2100 PRINT "   1  3  5  7  9 11 13 15"

```

FIGURE 7-6 LOWCOLOR Program.

Type NEW and enter the program into ADAM.

SAVE it with the name LOWCOLOR and RUN it. (see Figure 7-6)

Now you see the value you need to assign to COLOR to get colored squares you want to PLOT. You may have to adjust the tint or color knob on your television or monitor to match the names and numbers of colors in Figure 7-7.

<i>Number</i>	<i>Color</i>	<i>Number</i>	<i>Color</i>
0	Black	8	Light yellow
1	Magenta	9	Medium red
2	Dark blue	10	Gray-2
3	Dark red	11	Light red
4	Dark green	12	Light green
5	Gray-1	13	Light yellow
6	Medium green	14	Cyan (blue)
7	Light blue	15	White

FIGURE 7-7 Low-Resolution Color Values.

From your experience using SmartWriter and the BASIC line editor, you can list a few of the useful functions you want in the PICMAKER program.

You should come up with a list of functions similar to the following:

- Clear screen to black
- Move cursor up
- Move cursor down
- Move cursor left
- Move cursor right
- PLOT a color square on the screen at the cursor
- Erase a color square (set the color to black)
- Save the picture on cassette tape
- Load a picture from cassette tape
- Establish a color
- Exit from the program

Also consider these less obvious functions:

- Help—to explain how to use the program
- Brush on—paint a color wherever you move the cursor
- Brush off—just move cursor but don't paint color

THE PROCESSING TOP-DOWN DIAGRAM

Pictures, sketches and diagrams allow you to visualize your ideas. You will find drawing a diagram is especially useful in designing a program, such as PICMAKER, which consists of quite a few functions or subprocesses. Just like an architect draws diagrams called blueprints to design a house, you can use diagrams to help you design the process you want to program. Figure 7-9 (next page) shows a top-down diagram of the PICMAKER process. It shows the structure of the PICMAKER process from the top all the way down to simple subprocesses. At the top you see the main process. The row of boxes beneath it shows the lower-level subprocesses, such as moving the cursor up, that together form the main PICMAKER process. The lines connecting the boxes just visually reinforce the idea that the subprocesses are all part of the main process. Notice that lines connect the store/get subprocess box to two subprocess boxes on the third row, store and get. This illustrates that a subprocess often breaks down into smaller and simpler subprocesses.

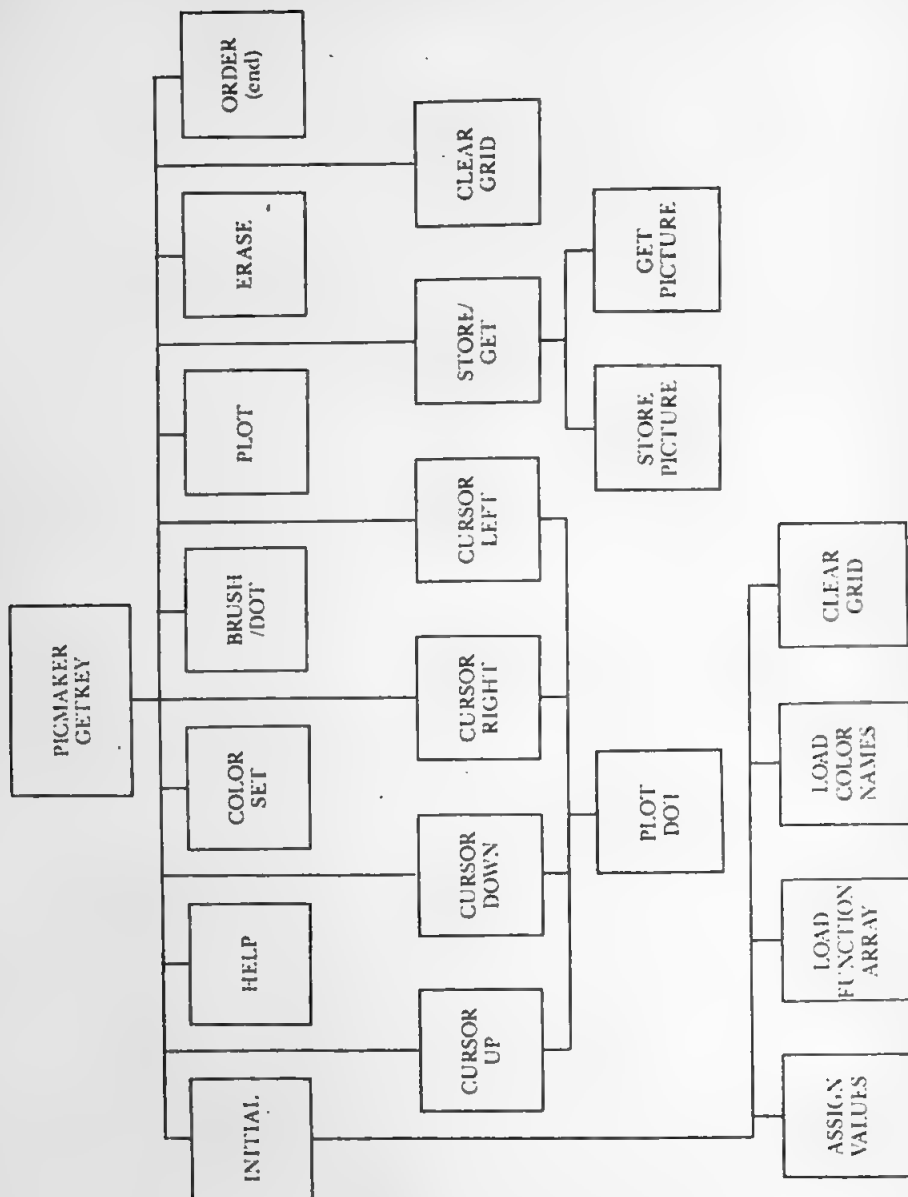


FIGURE 7-9 Top-Down Diagram of PICMAKER Program.

Now that you have a general idea how PICMAKER is designed, you can begin programming the main process.

THE MAIN PROCESS

The main process needs input from the keyboard to tell it what subprocess you want it to perform—where to place the square, what color, and how to store and get the picture on tape cassette.

You have used the INPUT command to read data from the keyboard into a variable in your program. It requires you to type letters or numbers and then press the RETURN key. But when you use SmartWriter to do cursor movements, you only have to press one of the arrow keys; other functions require the operator to press one of the function keys. You don't have to press the RETURN key.

The GET command allows you to do this in a BASIC program. It reads the value of the last key pressed into a string variable, but does not *echo* or show the character on the screen, so you don't see what you typed. You use a PRINT statement if you want to display the character you typed. Usually you want the function performed, not the character displayed. To see how it works type the following:

```
You type      GET(key$)
You type      A      (don't press RETURN)
You type      PRINT key$
ADAM displays A
```

Remember that ADAM represents letters with binary numbers. To get the decimal value of a character typed, you use the ASC function, which converts the first character of a string into the decimal value of its binary representation.

To continue the example:

```
You type      PRINT ASC(key$)
ADAM displays 65
```

Now type in the following program that will allow you to determine the value generated when you press any key on the keyboard. Use the same line numbers shown, since this little program will evolve into PICMAKER. Statement 1220 checks to determine if you pressed both the CONTROL key and the letter C. Normally, BASIC terminates the program when you press CONTROL-C, but with the GET statement it allows all key values to pass to the BASIC program. While the value of any key could have been used to terminate this program, choosing


```
10 REM  getkey
1000 REM  main loop
1020 PRINT "press any key!";
1100 GET key$
1220 IF ASC(key$) = 3 THEN STOP
1400 PRINT ASC(key$); TAB(20);
1410 IF ASC(key$) > 31 THEN PRINT key$
1420 PRINT
1990 GOTO 1000
```

FIGURE 7-10 GETKEY Program.

the conventional way makes it easier for you to remember. The STOP command ends the program with a message from BASIC. The PRINT command uses the TAB function to move to text column 20, where the character prints. The semicolon (;) after the TAB function prevents BASIC from starting the next PRINT statement on another line. This technique allows the key value and character to print on the same line.

We're going to turn the GETKEY program (Figure 7-10) into the PICMAKER program soon, so SAVE the program on cassette:

You type SAVE getkey

Now execute the program.

You type RUN

Appendix A lists the decimal values for all the keys, but you should try pushing the function keys, the cursor direction arrows, and the store/get and clear. Figure 7-11 shows these values. The PICMAKER program uses the values generated by pressing these keys to determine what subprocess you want it to perform.

To stop the program, press both the CONTROL key and the letter C (CTL-C).

ASCII values less than 32 won't print on the daisy wheel, but some display funny figures and do strange things. After trying this program a few times you may want to change statement 1410. Typing a statement with the same number replaces the existing statement in the program. Typing the number alone deletes the statement. To replace statement 1410:

You type 1410 PRINT key\$

You type RUN

Notice the face and eyes (oo). The video game in Chapter 8 uses these to represent the player game tokens.

<i>Value</i>	<i>Key pressed</i>
128	HOME
129	I
130	II
131	III
132	IV
133	V
134	VI
147	store/get
150	clear
160	up arrow
161	right arrow
162	down arrow
163	left arrow

Appendix A defines the values for all keys.

FIGURE 7-11 Function and Cursor Key Values.

FUNCTION OF THE KEYS

You tell SmartWriter what to do by using the cursor movement and function keys. Some of the function keys have words printed on them like CLEAR and STORE/GET to identify their function. The six general function keys ("smart keys") have Roman numerals, and SmartWriter puts prompts on the bottom of the screen to indicate their function. Since this technique makes it easy for the operator to choose what ADAM should do next, the picture-maker program will use it, too. Figure 7-12 (next page) shows how PICMAKER uses the four text lines available in the low-resolution graphics mode to identify the function keys with Roman numerals and display a word describing what the function key does. They are:

<i>Key</i>	<i>Prompt</i>	<i>Description</i>
I	Help	Lists the color names or describes the use of function keys. The <i>Companion</i> PICMAKER doesn't do this, but you can add these descriptions by using PRINT statements.
II	COLOR	Sets the color of the square that the plot key puts on the screen at the cursor location.

COLUMNS

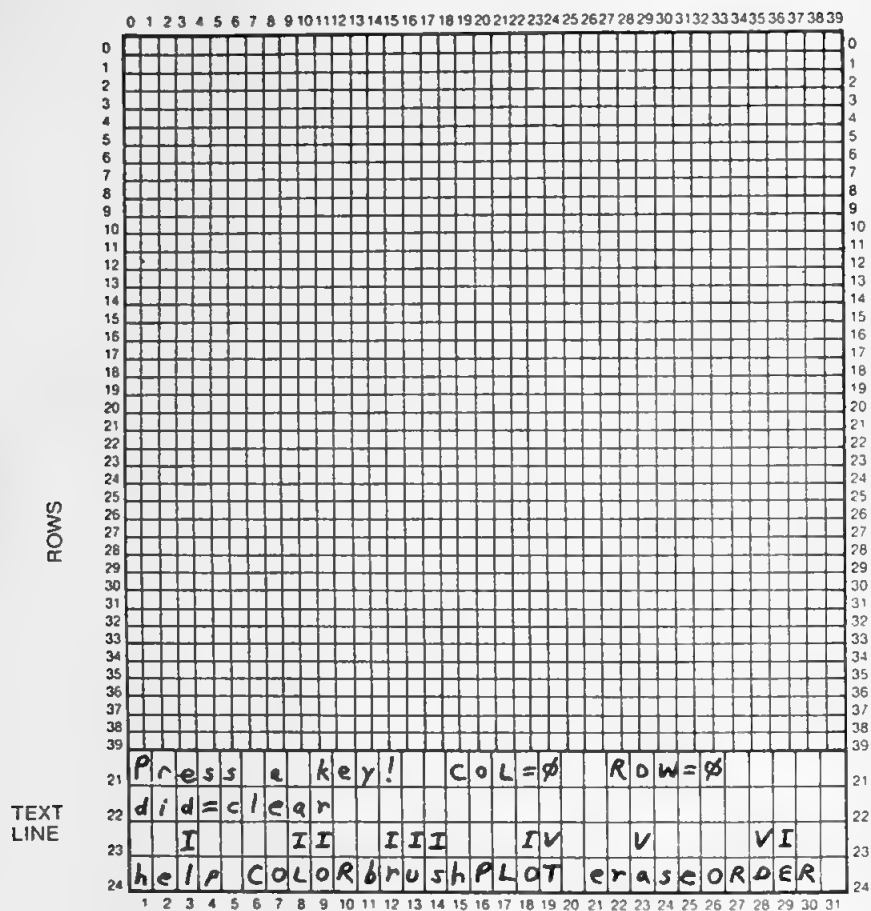


FIGURE 7-12 Identifying Function Keys.

- III Brush/dot In brush mode, the cursor acts like a paintbrush moving over the canvas. It puts colored squares wherever you move it. In dot mode, you need to press the PLOT function to put a colored square on the screen. Pressing the function key toggles back and forth between the brush and dot mode.
- IV PLOT Puts a colored square on the screen at the cursor location.
- V Erase Puts a black square on the screen at the cursor location.

- VI ORDER Allows you to enter an order or command, a word like *end*. This provides a way to add functions. You have to use words because we have assigned all the function keys. The program in the *Companion* supports only the *end* command but you may want to add others, like *shift*, that will shift the picture up or down on the screen.

Just like SmartWriter, the PICMAKER uses the arrow keys to move the cursor up, right, down, and left on the screen. The CLEAR key will color the screen black and put the cursor in the upper left-hand corner. The STORE/GET key will allow you to store or get a picture from the tape cassette.

STORING DATA IN ARRAYS

In the PICMAKER program you want to store the names of all 16 colors so the operator doesn't have to remember each color number but can enter a name like *red*, *green* or *yellow* and have PICMAKER convert the word to the number and then assign the number to COLOR. When you have a common collection of items, like the names of colors, instead of storing the names in 16 different string variables, you assign a name to the collection called an *array*. Figure 7-13 (next page) shows a diagram of an array of color names. You use a number as a subscript to reference or identify each item or element in the arrays. The value of a subscript can range from zero to the size of the collection. For example, the PICMAKER gives the name *shadeS* to the array of color names. The dollar sign means *shadeS* stores words or character strings. To reference the first element in the array you would write *shadeS(0)*. The number in the parentheses is the subscript. To reference the last element in the array you write *shadeS(15)*. Using a number greater than 15 would cause an error.

Let's experiment with arrays. String arrays are empty until you assign something to an element. To assign the word *black* to the first element in the array.

You type *shadeS(0) = "black"*

You type *?shadeS(0)*

ADAM displays *black*

If you don't define the size of the array, BASIC assumes a size of 10. To define a larger size, you need to use the DIMension command.

Once you define the size of an array, you can't change it. BASIC automatically dimensioned the array *shade\$* to 10 when you assigned the constant string "black" to the *shade\$*. Use the CLEAR command to clear data storage and then dimension the *shade\$* array to 15.

<i>Subscript</i>	<i>Element</i>
0	black
1	magenta
2	darkblue
3	darkred
4	darkgreen
5	gray-1
6	green
7	blue
8	vellow
9	red
10	gray
11	pink
12	palegreen
13	tan
14	cyan
15	white

FIGURE 7-13 Array of Color Names.


```
You type      CLEAR
You type      DIM shade$(15)
You type      shade$(15)="white"
You type      ?shade$(15)
ADAM displays  white
```

Notice that the subscript used for black and white is also the color number. If you load the *shade\$* array with the color names in order by color number, when you search the array for the entered color name and find a match you can use the subscript of the matched name as the value to assign to *COLOR*.

Arrays can have more than one dimension. You can have up to 88 dimensions in an array. The *PICMAKER* uses a two-dimensional array, named *grid%*, to store the color number of each colored square on the 40 column by 40 row low-resolution grid. To define it

```
You type      DIM grid%(39,39)
```

You should *DIMension* all arrays used by the program at the beginning of the program. This way, anybody looking at the program can easily find out what arrays the program uses. Also, it prevents the program from assigning values to the array before *BASIC* has processed the *DIMension* statement allocating the correct size of dimension to store the array.

USING THE DATA STATEMENT

A convenient way to load values into an array is to *READ* them from a *DATA* statement. Use the *DATA* statement to define a data list within a *BASIC* program. Use the *READ* statement to get a data item from the *DATA* list and assign it to a variable or element in an array. (Note: the *Companion* provides a complete definition and description of the *BASIC* language with examples and helpful hints in Appendix B. Please read it for more information about each *BASIC* statement and command.)

PICMAKER uses *DATA* statements to define the valid function key values. Each time the operator presses a function key *PICMAKER* must check that the value of the keys matches one of the functions provided by *PICMAKER*. If the operator presses the *WP* function key, *PICMAKER* will display an error message. Since it is faster to search an array than *READ* numbers from *DATA* statements, *PICMAKER* will *READ* the *DATA* into an array, named *func%*, during the initialization processes

and then search the array *func%* every time the GET command returns a key value. Type NEW and the following program:

```
10 REM PM7-14 1/21/84
100 DIM func%(12)
8500 REM initialize values
8600 DATA 160,161,162,163
8610 REM up,right,down,left
8620 DATA 129,130,131,132,133,134
8630 REM help,color,brush or dot,erase,order
8640 DATA 147,150
8650 REM store/get,clear
8670 FOR func = 1 TO 12
8680 READ keyvalue: func%(func) = keyvalue
8690 NEXT func
```

FIGURE 7-14 PICINIT Program.

Then SAVE the program with the name PM7-14. You will use it to merge with the GETKEY program to create the beginnings of PICMAKER.

RUN the program and observe that all 12 numbers get stored in the array *func%*.

You typed in statement 8685 so that you can check the value assigned to each element of the single-dimension array *func%*. It is just a way to assure the program works the way you want it to.

These statements are part of PICMAKER's initialization process. To combine them with the GETKEY program use SmartWriter to merge the two programs together. The line numbers in PM7-14 were selected so they wouldn't conflict with the numbers in GETKEY. Use SmartWriter to GET GETKEY first, then GET PM7-14. It doesn't matter if you don't merge them so the line numbers are in ascending order. SmartBASIC will put them in order when it reads the merged file. Use SmartWriter to delete 10 REM PM7-14 and statement 8685.

Then STORE the combined program with the name PICMAKER1. Follow the BASIC LOAD procedure and use the LOAD to bring the combined program PICMAKER1 into BASIC's program storage. Print the program and make sure it agrees with the listing in Figure 7-15. Use the BASIC line editor to correct any differences and SAVE the corrected program.


```

10 REM  picmaker1 1/21/84
100 DIM func%(12)
1000 REM  main loop
1020 PRINT "press any key!";
1100 GET key$
1220 IF ASC(key$) = 3 THEN STOP
1400 PRINT ASC(key$); TAB(20);
1410 IF ASC(key$) > 31 THEN PRINT key$
1420 PRINT
1990 GOTO 1000
8500 REM  initialize values
8600 DATA 160,161,162,163
8610 REM  up,right,down,left
8620 DATA 129,130,131,132,133,134
8630 REM  help,color,brush or dot,erase,order
8640 DATA 147,150
8650 REM  store/get,clear
8670 FOR func = 1 TO 12
8680 READ keyvalue: func%(func) = keyvalue
8690 NEXT func

```

FIGURE 7-15 PICMAKER1 Program.

GOSUB TO SUBROUTINES

The initialization statements now follow the main process. You want them executed once at the start of the program. How can you tell BASIC to execute these first and then come back and execute the main part of program? You use the GOSUB in conjunction with the RETURN statement. Type the following statements:

```

510 GOSUB 8500:REM do initialization
15000 RETURN

```

Figure 7-16 (next page) shows how the GOSUB transfers statement execution from statement 510 to statement 8500. BASIC saves the line number after statement 510. After the initialization routine executes, the RETURN statement at the end of the initialization routine causes BASIC to resume execution at the saved line number, 1020.

CHECKING THE KEYS

Now modify GETKEY to match the key value entered with those in the array. A *no-find condition* means that the operator pressed the wrong

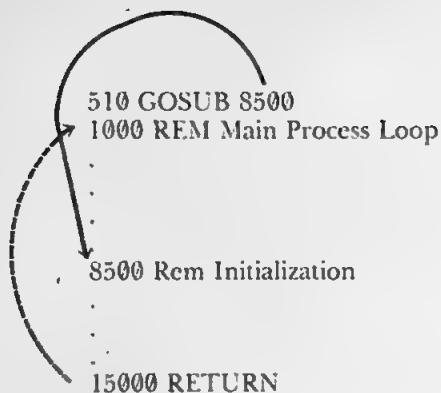


FIGURE 7-16 Diagram Illustrating GOSUB.

key. You want to tell the operator there's a mistake and allow another try. Enter the following statements:

```

1400 FOR func = 1 TO 12
1410 IF func%(func) = ASC(key$) THEN 1600
1420 NEXT func
1450 PRINT beep$; "WRONG KEY! PRESS FUNCTION KEY "
1550 GOTO 1000
1600 PRINT "VALID KEY="; ASC(key$)

```

FIGURE 7-17 Add to PICMAKER1.

CHRS(7) MAKES A BEEP SOUND

The ASC function converts a character to a number. The CHRS function converts a number to a character. The unprintable ASCII character seven makes a beep sound. Try it.

You type ?CHRS(7)

ADAM beeps

If you didn't hear it, turn up the volume on the TV. Add statement 8510 to make ADAM beep twice when the operator makes a mistake.

```
8510 beep$=CHRS(7)+CHRS(7)
```

RUN

Press all the 12 valid function keys and then try some letters and numbers. Listen to those beeps!

POSITIONING THE CURSOR

The screen layout for PICMAKER (Figure 7-12) shows that the error messages should always appear on line 22. The VTAB command allows you to position the cursor to any text line from 1 to 24. The HTAB command allows you to position the cursor to any text column from 1 to 31. Experiment!

You type HTAB 1:VTAB 22:?"ERROR"

ADAM displays ERROR at line 22

Mostly, you will use HTAB and VTAB together. For consistency with the PLOT command, which requires the column defined first, you should write the HTAB command to position the cursor to the column and then VTAB to position the cursor to the row.

Add statement 1440 to position the error message.

You type 1440 HTAB 1:VTAB 22

After adding these statements, to protect your investment in typing time, SAVE PICMAKER1 by typing:

You type SAVE picmaker1

PUTTING UP THE STUDS

With the PICMAKER1 program as a foundation you can build the framework of the PICMAKER program. For each subprocess add a REM statement to identify the subroutine that will perform the subprocess. Also, add a PRINT statement that prints the name of the subprocess and a RETURN statement. A GOSUBroutine command in the main process loop will evoke each subroutine, and the RETURN command assures that when complete the subroutine goes back to the main process. The PRINT statements will show the operator what function key was pressed and help you test the program. See Figure 7-18 (next page).

ON . . . GOSUB

Now that you have dummy subroutines in place you need a multiway switch to transfer execution to the subroutines when the operator presses the proper key. The ON . . . GOSUB command allows you to select a subroutine to execute from a list of statement line numbers by


```
2000 REM move cursor up
2100 PRINT "up"
2190 RETURN
2200 REM move cursor right
2300 PRINT "right"
2390 RETURN
2400 REM move cursor down
2500 PRINT "down"
2590 RETURN
2600 REM move cursor left
2700 PRINT "left"
2790 RETURN
2800 REM help - functions/colors
2900 PRINT "help"
2990 RETURN
3000 REM set color
3100 PRINT "color"
3190 RETURN
3200 REM brush/dot
3205 PRINT "brush/dot"
3390 RETURN
3400 REM plot color
3500 PRINT "plot color"
3590 RETURN
3600 REM erase-black square
3700 PRINT "erase"
3790 RETURN
4000 REM order/command
4100 PRINT "command=";
4190 RETURN
5000 REM store/get picture
5100 PRINT "store/get"
5590 RETURN
5600 REM new clean screen
5610 PRINT "clear"
5890 RETURN
```

FIGURE 7-18 Print Statements.

using a number. The . . . is the number, a value from 1 to the size of the list. Add the following statements to PICMAKER1.

```
1600 HTAB 1: VTAB 21: PRINT SPC(30): PRINT SPC(30);
1610 HTAB 1: VTAB 22: PRINT "did=";
1620 ON func GOSUB 2000, 2200, 2400, 2600, 2800, 3000,
3200, 3400, 3600, 4000, 5000, 5600
```

FIGURE 7-19 Add to PICMAKER1.

Depending on the value of the variable *func*, BASIC will transfer execution to one of the 12 subroutines.

The cursor-positioning commands move the cursor to where the PICMAKER wants the function name displayed.

SAVE the program and then RUN it.

When you press the proper function key, PICMAKER displays the name of the function in the lower left of the screen.

DISPLAYING THE PROMPTS

You want the CLEAR key to black out the screen by issuing the GR command. This also blacks out the text lines where PICMAKER displays the prompts. So the CLEAR subroutine should also display the prompts. Add the following PRINT statements (Figure 7-20) to the PICMAKER1 program so the bottom two lines look like the layout in Figure 7-12.

```
5610 GR: shade = 15: REM  white
5620 VTAB 23
5630 PRINT " I "; " II "; " III ";
5640 PRINT " IV "; " V "; " VI ";
5650 PRINT "help "; "COLOR"; brushdot$;
5660 PRINT "PLOT "; "erase"; "ORDER";
8530 brushdot$ = " dot "
8700 GOSUB 5600: REM  clear screen
```

FIGURE 7-20 Add to PICMAKER1.

SAVE and RUN the program. Press CTL-C to terminate execution. You should see the PICMAKER layout on your video screen.

MOVING THE CURSOR

The white box in the upper left-hand corner acts as the cursor, showing the current painting COLOR, marking the square where you can PLOT or ERASE the color. The variables *column* and *row* keep track of the cursor location. The BOXCOLOR holds the value of the screen color that the cursor color is temporarily replacing. Unless you BRUSH, PLOT or ERASE it, PICMAKER will redisplay BOXCOLOR at the current location before moving the cursor to another square on the screen.

To move the cursor you:

1. Assign COLOR=the value of the box color

2. PLOT the BOXCOLOR square replacing the cursor color.
3. Depending on the direction you want to move, check that movement does not move the cursor off the screen and then add or subtract 1 from either the *column* or *row* variable.
4. Get the BOXCOLOR of the current location from *grid%(column, row)*. Step 1 will assign this value to COLOR unless you change it by pressing the ERASE, PLOT, or BRUSH function keys.
5. Assign COLOR the current painting color held in the variable SHADE. The cursor square is always the current painting color.
6. PLOT a square the color of the painting color showing the current cursor location.

Since steps 1 and 2 are common to all cursor movements, PICMAKER puts them in a subroutine that each cursor subroutine invokes with a GOSUB 3800. Only step 3 is different for each cursor key, and each cursor subroutine will perform the proper instructions to change the *row* or *column*. Rather than including a separate subroutine to invoke after adjusting the *row* and *column*, PICMAKER puts steps 4–6 in the main routine.

Add the following statements.

```

100 DIM func%(12), grid%(39, 39)
1010 HTAB 1: VTAB 21
1020 PRINT "press a key!"; TAB(15); "COL="; column; TAB(22); "row="; row
1070 COLOR = shade: PLOT column, row
1110 boxcolor% = grid%(column, row)
1120 COLOR = boxcolor%: PLOT column, row
2020 GOSUB 3800
2040 IF row > 0 THEN row = row-1
2220 GOSUB 3800
2240 IF column < 39 THEN column = column+1
2420 GOSUB 3800
2440 IF row < 39 THEN row = row+1
2620 GOSUB 3800
2640 IF column > 0 THEN column = column-1
3800 REM          cursor movement
3820 IF erase% THEN boxcolor% = black%: GOTO 3840
3830 IF dot% OR brush% THEN boxcolor% = shade
3840 COLOR = boxcolor%: PLOT column, row
3860 grid%(column, row) = boxcolor%
3880 dot% = false%: erase% = false%
3890 RETURN
5800 REM clear grid
5810 HTAB 1: VTAB 21: PRINT "CLEARING GRID"
5820 FOR cg = 0 TO 39
5830 FOR rg = 0 TO 39
5840 grid%(cg, rg) = black
5850 NEXT rg: NEXT cg

```

FIGURE 7-21 Add to PICMAKER1.

You make the cursor move by changing the value of the *row* or *column* variable, checking first that you haven't exceeded the 40 by 40 grid boundaries. You can't have a *row* or *column* less than 0 or greater than 39. To move right or left, you add or subtract one to the *column* variable. To move up or down, you subtract or add one to the *row*. Then the PLOT statement you just added at 1070 displays the cursor with the current color.

CHECKPOINT—PICMAKER2

Change the program name in the REM in statement 10 to PICMAKER2. SAVE the program with this new name. RUN the program to test that you can move the cursor. If it doesn't work, list the program and compare the listing to Figure 7-22. Make the necessary changes until the two agree.

```

10 REM picmaker2
100 DIM func%(12), grid%(39, 39)
510 GOSUB 8500: REM initialization
1000 REM main loop
1010 HTAB 1: VTAB 21
1020 PRINT "press a key!"; TAB(15); "COL="; column; TAB(22); "row="; row
1070 COLOR = shade: PLOT column, row
1100 GET key$
1110 boxcolor% = grid%(column, row)
1120 COLOR = boxcolor%: PLOT column, row
1220 IF ASC(key$) = 3 THEN STOP
1400 FOR func = 1 TO 12
1410 IF func%(func) = ASC(key$) THEN 1600
1420 NEXT func
1440 HTAB 1: VTAB 22
1450 PRINT beep$: "WRONG KEY! PRESS FUNCTION KEY"
1550 GOTO 1000
1600 HTAB 1: VTAB 21: PRINT SPC(30): PRINT SPC(30):
1610 HTAB 1: VTAB 22: PRINT "did=";
1620 ON func GOSUB 2000, 2200, 2400, 2600, 2800, 3000, 3200, 3400, 3600, 4000,
5000, 5600
1990 GOTO 1000
2000 REM move cursor up
2020 GOSUB 3800
2040 IF row > 0 THEN row = row-1
2100 PRINT "up"
2190 RETURN
2200 REM move cursor right
2220 GOSUB 3800
2240 IF column < 39 THEN column = column+1
2300 PRINT "right"
2390 RETURN
2400 REM move cursor down
2420 GOSUB 3800
2440 IF row < 39 THEN row = row+1
2500 PRINT "down"
2590 RETURN
2600 REM move cursor left
2620 GOSUB 3800
2640 IF column > 0 THEN column = column-1
2700 PRINT "left"
2790 RETURN
2800 REM help - functions/colors

```



```

2950 PRINT "help"
2990 RETURN
3000 REM      set color
3100 PRINT "color"
3190 RETURN
3200 REM      brush/dot
3205 PRINT "brush/dot"
3390 RETURN
3400 REM      plot color
3500 PRINT "plot color"
3590 RETURN
3600 REM      erase-black square
3700 PRINT "erase"
3790 RETURN
3800 REM      cursor movement
3820 IF erase% THEN boxcolor% = black%: GOTO 3840
3830 IF dot% OR brush% THEN boxcolor% = shade
3840 COLOR = boxcolor%: PLOT column, row
3860 grid%(column, row) = boxcolor%
3880 dot% = false%: erase% = false%
3890 RETURN
4000 REM      order/command
4100 PRINT "command":
4190 RETURN
5000 REM      store/get picture
5100 PRINT "store/get"
5590 RETURN
5600 REM      new clean screen
5610 GR: shade = 15: REM      white
5620 VTAB 23
5630 PRINT " I "; " II "; " III ";
5640 PRINT " IV "; " V "; " VI ";
5650 PRINT "help "; "COLOR"; brushdot%;
5660 PRINT "PLOT "; "erase"; "ORDER";
##### REM      clear grid
5810 HTAB 1: VTAB 21: PRINT "CLEARING GRID"
5820 FOR cg = 0 TO 39
5830 FOR rg = 0 TO 39
5840 grid%(cg, rg) = black
5850 NEXT rg: NEXT cg
5890 RETURN
8500 REM      initialize values
8510 beep$ = CHR$(7)+CHR$(7)
8530 brushdot$ = " dot "
8600 DATA 160,161,162,163
8610 REM      up,right,down,left
8620 DATA 129,130,131,132,133,134
8630 REM      help,color,brush or dot,erase,order
8640 DATA 147,150
8650 REM      store/get,clear
8670 FOR func = 1 TO 12
8680 READ keyvalue: func%(func) = keyvalue
8690 NEXT func
8700 GOSUB 5600: REM      clear screen
15300 RETURN

```

FIGURE 7-22 PICMAKER2 Program.

Setting the Color

To change the paint color you need to change the value of the variable *shade*. Rather than have the operator remember the number for each color, PICMAKER allows the operator to enter *either* a color name or a number. If neither the name nor the number is valid, PICMAKER displays a message on line 22.

BAD COLOR NUMBER or WORD

You need to enter two routines. The first loads the *shade\$* array with the 16 color names. Add the following statements to the initialization subroutine:

```
100 DIM func%(12), grid%(39, 39), shade$(15)
9000 DATA "black"
9010 DATA "magenta"
9020 DATA "darkblue"
9030 DATA "darkred"
9040 DATA "darkgreen"
9050 DATA "grey-1"
9060 DATA "green"
9070 DATA "blue"
9080 DATA "yellow"
9090 DATA "red"
9100 DATA "grey"
9110 DATA "pink"
9120 DATA "palegreen"
9130 DATA "tan"
9140 DATA "cyan"
9150 DATA "white"
9200 REM      load color array
9210 FOR hue = 0 TO 15
9220 READ shade$(hue)
9250 NEXT hue
```

FIGURE 7-23 Setting the Color.

The second routine makes the SET COLOR subroutine do the following:

1. Display a prompt so the operator knows to enter a color.
2. Check to see if the operator entered a number between 0 and 15. If so, assign the value of the number, the new paint color value, to the variable *shade\$*.
3. If the operator did not enter a valid number, the FOR . . . NEXT loop searches the array of color names. If it finds a match, it sets *shade\$* to the value of *hue*, the subscript. The subscript becomes the current value of the current painting color, which is the cursor color.
4. If the word in *shade\$* doesn't match any of the color names in the *shade\$* array, display the error message.

Enter the following statements.

```

3010 PRINT "color"
3020 HTAB 1: VTAB 21: INPUT "enter color="; shade$
3030 IF LEN(shade$) > 1 THEN 3050
3040 IF shade$ < "0" OR shade$ > "9" THEN 3050
3045 GOTO 3060
3050 IF shade$ < "10" OR shade$ > "15" THEN 3100
3060 shade = VAL(shade$)
3080 RETURN
3100 REM          search for color name
3110 found% = false%
3120 FOR hue = 0 TO 15
3130 IF shade$(hue) = shade$ THEN shade = hue: found% = true%
3140 NEXT hue
3150 IF NOT found% THEN HTAB 1: VTAB 22: PRINT "BAD COLOR NUMBR or WORD"
8520 true% = 1: false% = 0

```

FIGURE 7-24 Setting the Color.

SAVE the program.

Plotting and Erasing the Color

To PLOT you assign *shade*, the current paint color, to BOXCOLOR. To ERASE you assign the value of the color black, zero, to BOXCOLOR. This is done in the routine you added at 3800. Here all you have to add is the statements to set the switches to true or false to activate the process. You will often use program switches to store the truth or falsity of a condition, like press ERASE function key. Rather than write *erase%=1* to assign a value one for true, you make the statement more understandable by assigning one to a variable named *true%* and then writing *erase%=true%*. Use the variable name *false%* to assign zero, which means false.

```

3420 dot% = true%: erase% = false%
3620 erase% = true%: dot% = false%

```

FIGURE 7-25 Plot Statements.

Brush and Dot

When you press the BRUSH/DOT function key you change the value of an integer variable *brush%* from true to false and display the current mode, brush or dot, on the screen.

Add the following statements.


```

3205 PRINT "brush/dot"
3210 IF brush% = true% THEN 3250
3220 brush% = true%: brushdot$ = "brush"
3230 GOTO 3260
3250 brush% = false%: brushdot$ = " dot "
3260 VTAB 24: HTAB 11: PRINT brushdot$;

```

FIGURE 7-26 Brush and Dot Statements.

Order

The order function key allows you to give orders or commands to PICMAKER. In this version, the only order you can give is END or *end*. After working with PICMAKER you may want to add other orders; these statements provide the facility for you to do it. Add the following statements.

```

4110 VTAB 21: HTAB 1: INPUT "command="; cmd$
4120 IF cmd$ = "end" THEN 20000
20000 REM end of program
20100 TEXT: PRINT "END of PICMAKER"
21999 END

```

FIGURE 7-27 Order Statements

PICMAKER

Change the name in statement 10 to PICMAKER and SAVE the program under that name.

RUN the program and try drawing lines and dots. Don't make it too elaborate because you still need to add the statements to perform the STORE/GET process, which allows you to STORE the picture as a file on the tape cassette and then GET it back from the tape and have PICMAKER display it.

STORE/GET THE PICTURE

Make sure you SAVED the PICMAKER because you will write a separate

program to test the STORE/GET. Then you will use SmartWriter to merge the STORE/GET statements into PICMAKER.

SmartWriter uses sequential files to store the letters and reports you type. BASIC uses *sequential files* to store your programs. A file consists of records. In a BASIC program file, each numbered statement is a *record*. You can divide records into units of information, called *fields*. A BASIC record consists of two fields: the line number and the commands. In a record written by a BASIC program, each variable is a field.

For simplicity, not speed, PICMAKER writes records having only one field or variable. The STORE process writes a sequential file of records, one for each of the 160 elements in the array *grid%* that hold the color value for each square on the video screen as shown in Figure 7-28. Grid elements for row 7, columns 10 through 20 are shown having the value for the color yellow. In the STOREGET program you'll write a FOR . . . NEXT loop to load those values into the *grid%* array, simulating what the current version of PICMAKER does, to test that the program works.

<i>Record</i>	<i>Source</i>	<i>Value</i>	<i>Meaning</i>
1	grid%(0,0)	0	black
2	grid%(1,0)	0	black
	.		
40	grid%(39,0)	0	black
41	grid%(0,1)	0	black
42	grid%(1,1)	0	black
	.		
251	grid%(10,7)	8	yellow
252	grid%(11,7)	8	yellow
253	grid%(12,7)	8	yellow
	.		
261	grid%(20,7)	8	yellow
262	grid%(21,7)	0	black
	.		
2400	grid%(39,39)	0	black

FIGURE 7-28 Records Written and Read by STOREGET.

The GET process reads the file and reloads the *grid%* array. It uses the value of the *grid%* element defined by the *column* and *row* variables to assign COLOR. Then it uses the PLOT command to place a colored square at the *column* and *row* screen location.

To create a sequential file you need to use four commands:

OPEN—Defines the filename and tape drive.

WRITE—Defines the operation as opposed to READ. It tells BASIC to direct all PRINT statement data to the tape instead of the video screen.

PRINT—Writes a record consisting of the variables in the print list. It formats the variables just as it would if you were to see it on the screen, except it writes the data to the tape instead of the screen.

CLOSE—Tells BASIC you finished PRINTing to the tape and resumes displaying PRINT statement data on the screen.

To maintain compatibility with Applesoft BASIC, SmartBASIC requires you to use the PRINT statement to pass file commands to it. To tell BASIC that you are giving it a file command and not data, the first variable in the PRINT list must have the ASCII character value 4, which you define with CHR\$(4). If you PRINT CHR\$(4) all by itself it displays as a heart. Try it.

You type ?CHR\$(4)

ADAM displays a heart

Happy Valentine's day!

By convention, because it takes up less space in the statement line, the character string variable DS is assigned the ASCII value CHR\$(4). Then DS rather than CHR\$(4) is used as first character in the PRINT list passing a file command. Type up the following program which creates a one-record file named HEART. We'll keep modifying this program until you have written the STORE/GET function for PIC-MAKER.

```

10 REM   storeget
110 name$ = "HEART"
120 d$ = CHR$(4)
5300 REM   save picture
5340 PRINT d$; "open "; name$; ",d1"
5350 PRINT d$; "write "; name$
5370 PRINT "have a heart"
5385 PRINT d$; "close "; name$
5387 HTAB 1: VTAB 22: PRINT "PICTURE "; name$; " SAVED"
```

FIGURE 7-29 STOREGET.

SAVE the program with the name STOREGET.

Want to watch BASIC do the file operations? Use the MON command to turn the monitor switches on. The C means display commands and the O means display output. To see any input use the letter I. Use NOMON with the proper letter to turn the monitor switches off.

You type MON C,O

Now execute the program.

You type RUN

ADAM displays open HEART,D1
 write HEART
 have a HEART
 close HEART
 PICTURE HEART SAVED

 You can use SmartWriter to look at the record you
 w n use SmartWriter to change the contents of a
 fi recommends you use the window display mode
 w rds containing more than 70 characters. You can
 r the contents of a file, but be careful.
 sor, not a text editor. It has routines
 ing printout and therefore may change
 the record.

READING THE FILE

To read a sequential file, use the READ command instead of WRITE. It tells BASIC to get all data for the INPUT command from the tape cassette and *not* the keyboard. When you CLOSE the file, all INPUT again comes from the keyboard.

Make sure you are in BASIC and have the small STOREGET program LOADED into BASIC program memory. Add the following statements, which allow you to enter a filename and read a single record. The LENGTH function returns the length of a string variable. If the operator just pressed the RETURN key in response for a picture name, the length of the string will be zero.

After all that typing, SAVE the program (Figure 7-30).

Turn the monitor on to look at commands, input and output.


```

900 GOSUB 5000
950 IF none% THEN 900
960 END
1000 GOSUB 5300: REM          save picture
1050 GR: VTAB 23: HTAB 1: PRINT "loading file";
1100 GOSUB 5200: REM          load picture
1200 END
5000 REM          store/get picture
5100 PRINT "store/get"
5110 HTAB 1: VTAB 21: INPUT "STORE/GET(s/g)?"; ans$
5120 IF ans$ = "S" OR ans$ = "s" THEN 5300
5130 IF ans$ = "G" OR ans$ = "g" THEN 5200
5140 HTAB 1: VTAB 22: PRINT "Not s or g": RETURN
5200 REM          load picture
5210 op$ = "get "
5220 GOSUB 5550: IF none% THEN RETURN
5240 PRINT d$; "open "; name$
5250 PRINT d$; "read "; name$
5270 INPUT record$
5282 PRINT d$; "close "; name$
5284 RETURN
5550 REM          input picture name
5555 none% = false%
5557 HTAB 1: VTAB 21: PRINT SPC(30);
5558 HTAB 1: VTAB 21: PRINT op$;
5560 INPUT "picture name="; name$
5570 IF LEN(name$) > 0 THEN RETURN
5580 HTAB 1: VTAB 22: PRINT "NO PICTURE "; name$
5585 none% = true%
5590 RETURN

```

FIGURE 7-30 STOREGET Sample Output.

You type MON C,I,O

You type RUN

When the program displays

STORE/GET(s/g)?

You type g (to get file)

When the program displays

Picture name=

You type HEART (in capital letters)

The cassette drive hums and cranks as it updates the directory, changing the entry for the first *HEART* file to a backup, writing a new entry, writing a new file, going back to the directory to update for the last block used by the new *HEART* file, then reading the directory again when the program opens the file for reading, positioning tape to the first block in the file, then reading the first record. You can watch all the action on the screen.

FILE NOT FOUND

Suppose you didn't spell the filename *HEART* correctly. When BASIC can't find the filename you entered, it displays the error message:

FILE NOT FOUND

and terminates the program. This is not what you want to happen. The program should inform you that it can't find the file and give you another chance to enter the correct picture filename. Enter the following statements.

```
5230 ONERR GOTO 5285
5285 REM      onerr routine
5286 CLRERR: GOSUB 5580
5290 PRINT d$; "close "; name$
5293 PRINT d$; "delete "; name$
5299 RETURN
```

FIGURE 7-31 ONERR.

The ONERR command allows you to override BASIC normal error-handling routines with your own. The CLRERR command cancels the ONERR command, letting BASIC handle errors. You only want to handle a FILE NOT FOUND error, so place the ONERR command just before the OPEN. The error-handling routine executes the CLRERR command to prevent BASIC from looping in the ONERR routine. It displays the NO PICTURE error message. It CLOSEs the file that was created by the OPEN command when it couldn't find the filename. The DELETE command removes the erroneous filename from the directory. Finally it RETURNs to statement 950. If the *none%* switch is true, the INPUT prompt routine is executed again. In PICMAKER you will press the STORE/GET key to reenter the filename again.

SAVING THE GRID

Now that the STORE/GET program writes and reads one record, you can add a few simple additional statements (Figure 7-32) to write and read the whole *grid%* array. Nested within a FOR . . . NEXT loop for columns, the FOR . . . NEXT loop for rows writes or reads the records for each element in the array *grid%*.


```

5260 FOR cg = 0 TO 39
5265 FOR rg = 0 TO 39
5270 INPUT " "; grid%(cg, rg): CLRERR
5280 NEXT rg: NEXT cg
5360 FOR cg = 0 TO 39
5365 FOR rg = 0 TO 39
5370 PRINT grid%(cg, rg)
5380 NEXT rg: NEXT cg

```

FIGURE 7-32 Add to STOREGET.

In the middle of the READ row FOR...NEXT loop places statements to assign the COLOR and PLOT the square.

```
5275 COLOR=grid%(cg,rg):PLOT cg,rg
```

Add a DIMension statement to define the size of the *grid%* array.

```
100 DIM grid%(39,39)
```

Add statements to assign values to elements in the *grid%* that define a blue horizontal line from column 10 to 20 at row 20.

```

200 GR
300 FOR cg = 10 TO 20
310 grid%(cg, 20) = 7
320 NEXT cg

```

FIGURE 7-33 Add to STOREGET.

Finally, replace 900 with statements to just STORE and GET the *grid%* array.

```
900 GOSUB 5300
```

```
910 GR:VTAB 23:htab 1:?"loading file";
```

```
920 GOSUB 5200
```

SAVE it and RUN it.

In preparation for merging STOREGET with PICMAKER, use BASIC's line DELete command to delete the lines that tested the STORE/GET subroutines.

You type DEL 10,950

You type SAVE storegetm

MERGING THE TWO PROGRAMS

Use SmartWriter to MERGE the two programs. GET PICMAKER first, then use the search function to position SmartWriter to the

characters 5000 REM. Then GET STOREGETM. SAVE the program as PICMAKER. Insert the BASIC cassette, hit reset, and when BASIC is loaded put the PICMAKER tape back in. Load PICMAKER, then print it. You type

pr#1:list:pr#0

Check that the printout matches Figure 7-34. If it does, you're in business!

```

10 REM  picmaker 01/21/84
20 REM  copyright 1984 by Ramsey J Benson
100 DIM func%(12), grid%(39, 39), shade$(15)
510 GOSUB 8500: REM  initialization
1000 REM  main loop
1010 VTAB 21: HTAB 1
1020 PRINT "press a key!"; TAB(15); "COL="; column; TAB(22); "row="; row
1070 COLOR = shade: PLOT column, row
1100 GET key$
1110 boxcolor% = grid%(column, row)
1120 COLOR = boxcolor%: PLOT column, row
1220 IF ASC(key$) = 3 THEN STOP
1400 FOR func = 1 TO 12
1410 IF func%(func) = ASC(key$) THEN 1600
1420 NEXT func
1440 HTAB 1: VTAB 22
1450 PRINT beep$; "WRONG KEY! PRESS FUNCTION KEY"
1550 GOTO 1000
1600 HTAB 1: VTAB 21: PRINT SPC(30): PRINT SPC(30);
1610 HTAB 1: VTAB 22: PRINT "did=";
1620 ON func GOSUB 2000, 2200, 2400, 2600, 2800, 3000, 3200, 3400, 3600, 4000,
5000, 5600
1990 GOTO 1000
2000 REM          move cursor up
2020 GOSUB 3800
2040 IF row > 0 THEN row = row-1
2100 PRINT "up"
2190 RETURN
2200 REM          move cursor right
2220 GOSUB 3800
2240 IF column < 39 THEN column = column+1
2300 PRINT "right"
2390 RETURN
2400 REM          move cursor down
2420 GOSUB 3800
2440 IF row < 39 THEN row = row+1
2500 PRINT "down"
2590 RETURN
2600 REM          move cursor left
2620 GOSUB 3800
2640 IF column > 0 THEN column = column-1
2700 PRINT "left"
2790 RETURN
2800 REM          help - functions/colors
2900 PRINT "help"
2990 RETURN
3000 REM          set color
3010 PRINT "color"
3020 HTAB 1: VTAB 21: INPUT "enter color="; shade$
3030 IF LEN(shade$) > 1 THEN 3050
3040 IF shade$ < "0" OR shade$ > "9" THEN 3050
3045 GOTO 3060
3050 IF shade$ < "10" OR shade$ > "15" THEN 3100
3060 shade = VAL(shade$)
3070 RETURN
3100 REM          search for color name
3110 found% = false%
3120 FOR hue = 0 TO 15
3130 IF shade$(hue) = shade$ THEN shade = hue: found% = true%
3140 NEXT hue
3150 IF NOT found% THEN HTAB 1: VTAB 22: PRINT "BAD COLOR NUMBER or WORD"
3190 RETURN

```



```

3200 REM          brush/dot
3205 PRINT "brush/dot"
3210 IF brush% = true% THEN 3250
3220 brush% = true%: brushdot$ = "brush"
3230 GOTO 3260
3250 brush% = false%: brushdot$ = " dot "
3260 VTAB 24: HTAB 11: PRINT brushdot$:
3390 RETURN
3400 REM          plot color
3420 dot% = true%: erase% = false%
3500 PRINT "plot color"
3590 RETURN
3600 REM          erase-black square
3620 erase% = true%: dot% = false%
3700 PRINT "erase"
3790 RETURN
3800 REM          cursor movement
3820 IF erase% THEN boxcolor% = black%: GOTO 3840
3830 IF dot% OR brush% THEN boxcolor% = shade
3840 COLOR = boxcolor%: PLOT column, row
3860 grid%(column, row) = boxcolor%
3880 dot% = false%: erase% = false%
3890 RETURN
4000 REM          order/command
4100 PRINT "command"
4110 VTAB 21: HTAB 1: INPUT "command=": cmd$
4120 IF cmd$ = "end" THEN 20000
4190 RETURN
5000 REM          store/get picture
5100 PRINT "store/get"
5110 HTAB 1: VTAB 21: INPUT "STORE/GET(s/g)?": ans$
5120 IF ans$ = "s" OR ans$ = "S" THEN 5300
5130 IF ans$ = "g" OR ans$ = "G" THEN 5200
5140 HTAB 1: VTAB 22: PRINT "Not s or g": RETURN
5200 REM          load picture
5210 op$ = "get "
5220 GOSUB 5550: IF none% THEN RETURN
5240 PRINT d$: "open ": name$
5250 PRINT d$: "read ": name$
5260 FOR cg = 0 TO 39
5265 FOR rg = 0 TO 39
5270 INPUT " ": grid%(cg, rg): CLRERR
5275 COLOR = grid%(cg, rg): PLOT cg, rg
5280 NEXT rg: NEXT cg
5282 PRINT d$: "close ": name$
5284 RETURN
5285 REM          ONERR ROUTINE
5286 CLRERR: GOSUB 5580
5290 PRINT d$: "close ": name$
5293 PRINT d$: "delete ": name$
5299 RETURN
5300 REM          save picture
5310 op$ = "store "
5320 GOSUB 5550: IF none% THEN RETURN
5340 PRINT d$: "open ": name$
5350 PRINT d$: "write ": name$
5360 FOR cg = 0 TO 39
5365 FOR rg = 0 TO 39
5370 PRINT grid%(cg, rg)
5380 NEXT rg: NEXT cg
5385 PRINT d$: "close ": name$
5387 HTAB 1: VTAB 22: PRINT "PICTURE ": name$: " SAVED":
5390 RETURN
5550 REM          input picture name
5555 none% = false%
5557 HTAB 1: VTAB 21: PRINT SPC(30):
5558 HTAB 1: VTAB 21: PRINT op$:
5560 INPUT "picture name=": name$
5570 IF LEN(name$) > 0 THEN RETURN
5580 HTAB 1: VTAB 22: PRINT "NO PICTURE ": name$
5585 none% = true%
5590 RETURN
5600 REM          new clean screen
5610 GR: shade = 15: REM white
5620 VTAB 23
5630 PRINT " I ": " II ": " III ":
5640 PRINT " IV ": " V ": " VI ":
5650 PRINT "help ": "COLOR": brushdot$:

```



```

5660 PRINT "PLOT "; "erase"; "ORDER";
5800 REM   CLEAR GRID
5810 HTAB 1: VTAB 21: PRINT "CLEARING GRID";
5820 FOR cg = 0 TO 39
5830 FOR rg = 0 TO 39
5840 grid$(cg, rg) = black
5850 NEXT rg: NEXT cg
5890 RETURN
8500 REM           initialize value
8510 beep$ = CHR$(7)+CHR$(7): d$ = CHR$(4)
8520 true$ = 1: false$ = 0
8530 brushdot$ = " dot "
8600 DATA          160,161,162,163
8610 REM           up,right,down,left
8620 DATA          129,130,131,132,133,134
8630 REM           help,color,brush or dot,erase,order
8640 DATA          147,150
8650 REM           store/get,clear
8670 FOR func = 1 TO 12
8680 READ keyvalue: funct(func) = keyvalue
8690 NEXT func
8700 GOSUB 5600: REM   clear screen
9000 DATA "black"
9010 DATA "magenta"
9020 DATA "darkblue"
9030 DATA "darkred"
9040 DATA "darkgreen"
9050 DATA "grey-1"
9060 DATA "green"
9070 DATA "blue"
9080 DATA "yellow"
9090 DATA "red"
9100 DATA "grey"
9110 DATA "pink"
9120 DATA "palegreen"
9130 DATA "tan"
9140 DATA "cyan"
9150 DATA "white"
9200 REM           load color array
9210 FOR hue = 0 TO 15
9220 READ shade$(hue)
9250 NEXT hue
15000 RETURN
20000 REM           end of program
20100 TEXT: PRINT "END of PICMAKER"
21999 END

```

FIGURE 7-34 Final Listing of PICMAKER.

HAVE FUN

Let the family have fun drawing pictures. Copy PICMAKER to another tape cassette by loading and saving the program. Let the kids play with the copy, and save the original tape cassette in a safe, temperature-controlled place like your desk.

IMPROVEMENTS

You might consider these projects to test your newly acquired programming skills.

- Shorten the size of file by using condensing techniques to reduce

the file space required to save the picture and the time required to load it. For example, you need only four bits to define the 16 color values. Place two color values into the eight low-order bits of an integer and POKE the byte into a memory location below LOMEM:. Then use the BSAVE command to save a file containing the bytes that describe the pictures. Appendix B describes how to use these commands.

- Write commands that allow you to move or shift picture from one position to another on the screen.
- Write a command to save the picture in another *grid%* memory array so you can quickly flip back and forth between pictures.
- Allow pictures to be merged.
- Display a picture at the beginning of PICMAKER to announce or title it.
- Modify STOREGET to become a slide-show displayer, displaying one picture file after another.

Writing Your First Video Game in BASIC

What makes a good video game? Why do we enjoy playing Mousetrap, Carnival, Donkey Kong and Zaxxon? Consider each game a process, just like the pocket calculator or drawing a picture, that you must study to see how they work and what features make them successful.

The *Companion* played all of Coleco's popular games and concluded that an enjoyable video game has most or all of the features below:

- Visual action enhanced by the appropriate sounds or music.
- Simplicity—so most people can master the basic skills needed to play the game.
- Levels of difficulty—players of the game will have various levels of intelligence and skill. To appeal to many people the game should have levels of difficulty. With practice, many game fanatics will master the first level. Providing higher levels of difficulty extends the enjoyable playing life of the game.
- Satisfaction—each game gives the player gratification in terms of points as a reward for successfully accomplishing a feat. Satisfaction comes from meeting a challenge, solving the puzzle or doing the task.
- Surprise or discovery—the thrill of learning something new or solving a puzzle gives adventure games their compelling appeal.
- Fantasy—the games allow us to live out fantasies that would otherwise be denied us.

IDEAS FOR GAMES

You can get a game idea any place. Look up in the sky at the clouds or down in the grass, look at animals, insects or your own activities. Use

your imagination! Slither has the player shooting snakes in the desert. Listen to a song, watch a movie or television show, read a book—the dictionary or encyclopedia is filled with words and topics that can trigger an idea for a game. Take any idea and make it into a game. Soap bubbles going down the drain became an arcade game called Bubbles. Maybe you can dream up a game about bubble gum bubbles!

You can classify games into various types:

Action games—Donkey Kong, Rocky, Baseball

Shoot-'em-up games—Carnival, Slither, Buck Rogers

Maze games—Mousetrap, Ladybug

Card games—Blackjack

Adventure games—Venture or Smurf

Board games—Monopoly, chess and backgammon

Educational games—Scrabble and other word and number games

Once you get your idea, see if you can design it so it fits into one of these categories or contains combinations of these categories. Maybe you can think up a new category and design a game for it.

Make sure you design the game to incorporate the features of others that have proved successful. Is it simple to play? Does it give the player a reward for successful playing? Are there enough surprises and thrills to keep the player interested in playing the game? Get your family and friends to review your design and help you improve it. While it's fun, coming up with a good game design takes some effort, time and thought.

GAME-PLAYING EQUIPMENT

As you design your game, take into consideration the equipment you will use to play it. ADAM has two hand controllers and a keyboard that the player can use to input the action instructions. With the controllers you can move Donkey Kong up the ladders or fire a bullet at the enemy warship. Many popular adventure games, like Princess and the Whizard, use the keyboard to get the player's directions, like GO NORTH or ENTER ROOM. The keyboard's function keys provide an alternate method of selecting choices that your game may offer the player.

how to use the PDL function to read the controller values that the player generates by pressing the buttons or keypad or pushing the joystick. You can use these values in your game to determine the number of players or when to fire, move, etc.

You display the action of the game on the television screen using one of SmartBASIC's three video display modes—TEXT, low resolution and high resolution. You write BASIC programs in the TEXT mode, but as you'll soon learn, it can also function as game board. The PICMAKER uses the *low-resolution* mode to let you draw pictures using small colored squares. You can create many exciting games incorporating the pictures you draw with PICMAKER. The *high-resolution* mode uses dots instead of squares, so you can create a much more detailed figure. You define a video figure, like a star, using bits and bytes. These bits and bytes definitions get stored in a *shape table*, which you can reference with BASIC commands. It's complicated and takes a lot of time to design the figures and convert them into a shape table. If you want to try using high-resolution graphics, Appendix D explains how. All the Coleco games use a special Z-80 microprocessor version of the high-resolution mode similar to SmartBASIC's but much faster.

GAME CASE STUDY: CATCH A FALLING STAR

The *Companion's* first idea for a game came from a song made popular by Perry Como in the early 1960s called "Catch a Falling Star." Stars numbered from 1 to 9 fall continuously from the top of the video screen. The player uses the joystick to control a basket at the bottom of the screen to catch the falling stars until the total shown on the front of the basket equals a target total, say 100. The prototype model used a shape table to store the stars, basket and numbers. However, using the high-resolution graphics DRAW and XDRAW BASIC commands to move the stars and basket around the screen proved too slow for the game to challenge the average player's skills. The *Companion* concludes that while great for prototyping, BASIC is not a satisfactory language for developing high-resolution, fast-action games using DRAW and XDRAW to move the video figures quickly. In the future, Coleco will introduce an enhanced BASIC so you can quickly move video figures around on the screen.

Don't be disappointed; you can still program lots of low-resolution

games in BASIC that are tons of fun. If you want to use high-resolution graphics, just design the game so the figures stay still or move slowly.

LOW-RESOLUTION GAMES

Some excellent games have been written using the Apple computer low-resolution graphics mode. SmartBASIC's low-resolution mode is almost identical to it. Both use a 40 column by 40 row grid of colored squares, 4 text lines and the same BASIC commands. Apple's text lines are 40 characters in width; while the SmartBASIC text lines are 31.

Applesoft and SmartBASIC are not identical. SmartBASIC lacks Apple's many computer-dependent PEEKs and POKEs. For example, the Apple has a built-in speaker that you click by using PEEK(49200); this won't work in SmartBASIC. ADAM uses the speaker in your television to create sound. Chapter 9 explains how to create sounds and music with ADAM. The *Companion* expects that Coleco will publish ADAM's own useful memory locations. If you type a published game program written in Applesoft BASIC, just remember that except for shape-table POKEs, don't include the PEEKs and POKEs. Some Applesoft POKEs may even cause ADAM to stall. If that happens you will have to RESET and reLOAD SmartBASIC. Before using shape-table POKEs, read Appendix D and what Appendix B says about the HIMEM: and LOMEM: commands.

One of the most famous low-resolution games, called both Breakout and Brickout, similar to handball or racketball, involves using the controller as an electronic paddle (a vertical line of 4 squares) to bang a ball (a white square) against a colored brick wall 10 layers or squares thick. Bricks have a point value from 1 to 10, the closer layer having the lower value and the right-hand layer the highest. When the ball hits a brick, it destroys it and the player wins the point value of the brick. The ball bounces back and the player must move the paddle to hit and return the ball. If the player misses, he or she loses a chance and has three chances to run up a good score. Maybe you could write a version for ADAM!

Maze games and shoot-'em-up action games, such as Pac-Man and Space Invaders, work very well using the colorful low-resolution video display mode.

ADAM'S COMPANION

LETTER CHASER

To illustrate some of the techniques of writing a game program, the *Companion* created the Letter Chaser. The inspiration came from a combination of playing Mr. Do, looking up how to spell words in a dictionary and testing out what ADAM displays for each of the 256 character values.

It uses the TEXT mode, displaying a forest of letters randomly on the screen (see Figure 8-1). Like a reverse Scrabble, players use the joystick to chase through the forest trying to spell the word displayed on the bottom of the screen. The game displays a target word with dashes underneath. When the player moves the token to a matching letter, the letter pops into the dash under the target letter, marking the player's progress. To start the search again the player must push either the left or right joystick button. Trying to move without a button push sounds a beep. To add an element of strategy to the game, as the player moves out of a square, the letter disappears. So as the player moves a maze is created.

The real excitement comes in a two-player game when each player uses a controller to race around the screen trying to complete the target word first and blanking out letters the opponent needs at the same time.

The game ends when the player matches all the letters. As a surprise and reward, Letter Chaser displays a colorful, low-resolution kaleidoscope.

The game should help all of us, especially 8-12-year-olds, to develop our spelling skill. The program reads the words from a file named *chaseword* into array, just as PICMAKER reads the value for each color in the picture grid. While BASIC must create the file with at least one word, you can use SmartWRITER to add to and change the file of words, one word per line or record.

Letter Chaser Design

Figure 8-2 (p. 172) diagrams the overall design of the LETTRCHASE program. As we've explained, filenames can be no more than ten characters long; so Letter Chaser becomes LETTRCHASE. The top box represents the main process that reads the paddles and then invokes the subprocesses in the boxes below it to implement the game. Study the diagram; do you see the similarities to what you learned in

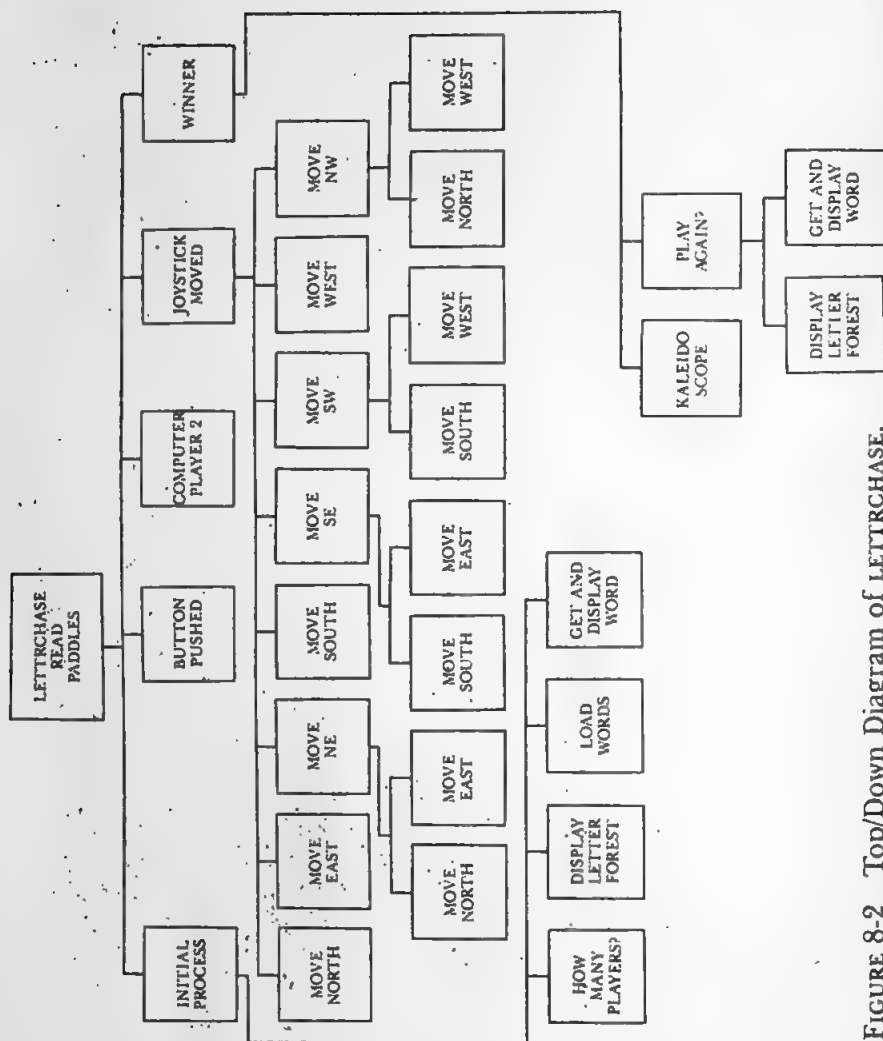


FIGURE 8-2 Top/Down Diagram of LETTRCHASE.

Tokens

The Letter Chaser game uses tokens, graphic symbols, as a cursor to mark the player's location in the forest of letters. While exploring SmartBASIC, the *Companion* discovered that the PRINT command displays, on the video screen, a face for CHR\$(2) and twin circles or eyes for CHR\$(15). See for yourself.

You type: ?chr\$(2);chr\$(15)

Adam displays: snake eyes and a face

Appendix A shows other symbols that you may want to use in your games. Because words are more meaningful than numbers, LETTRCHASE assigns the CHR\$(2) values to the string variables *face\$* and *eyes\$*.

Storing Two-Player Information

the LETTRCHASE program uses string and integer arrays to store the information common to both players. For example, in the initialization routine the *face\$* is assigned to *player\$(1)* and *eyes\$* is assigned to *player\$(2)*. The integer variable *player%* subscripts each array, pointing to the information for the player. For example, *player\$(player%)* refers to the token value for either player (controller) depending on whether *player%* has a value one or two. Many of the subroutines, like moving the tokens, perform the same process for either player depending on the value of the variable *player%*.

The game uses integer arrays, array names with the percent sign (%), rather than decimal arrays, because BASIC performs integer arithmetic much faster than decimal arithmetic. Response time in video games is important to the player. When the player pushes the joystick, the token must move quickly, otherwise the player becomes frustrated and doesn't want to play the game. The time it takes BASIC to execute instructions to move the cursor determines the response time. With decimal arithmetic BASIC must keep track of the decimal point, and that takes a lot of time. To get the best response time in your games, wherever possible use integers to store the game values.

The Main Loop

Like PICMAKER and most game programs, this program has a major processing loop that reads a value supplied by the operator. The

value determines which subprocess, such as cursor movement, the program must perform. Here the operator uses the hand controller, instead of the cursor and function keys, to send information to the program.

Type up the following program, which uses the PDL function to read the value of the controllers. If any PDL function returns a nonzero or true value, the program prints a line showing the value of the controller function. To see what happens when you move the controller joystick, press either the left or right button, or press a keypad square. It does not show the paddle wheel controller values since *LETTRCHASE* doesn't use them. Gradually you will add routines to this program to turn it into the Letter Chaser game, so SAVE it before RUNning it. Name it *READPDL*. The integer arrays *joy%* and *key%* store the joystick direction and keypad values of each of the controllers. The integer variables *l1%* and *l2%* store the left-button values, and *r1%* and *r2%* the right-button values. Pressing the asterisk on the controller keypad terminates the program.

```

      REM readpdl
1 . DIM joy%(2), key%(2), move%(2), row%(2), column%(2)
1030 REM      main game process
1020 COSUB 8000
2000 REM      read controllers
2020 joy%(1) = PDL(5): REM      joystick 1
2040 joy%(2) = PDL(4): REM      joystick 2
2060 key%(1) = PDL(11): REM     keypad 1
2080 key%(2) = PDL(10): REM     keypad 2
2100 IF key%(1) = sign% OR key%(2) = sign% THEN TEXT: END
2120 IF key%(1) = asterisk% OR key%(2) = asterisk% THEN 1000
2200 l1% = PDL(7): REM      left button 1
2210 l2% = PDL(6): REM      left button 2
2220 r1% = PDL(9): REM      right button 1
2230 r2% = PDL(8): REM      right button 2
2240 IF NOT (joy%(1)+key%(1)+l1%+r1%) THEN 2255
2250 PRINT "1 joy="; joy%(1); ",key="; key%(1); ",left="; l1%; ",right="; r1%
2255 IF NOT (joy%(2)+key%(2)+l2%+r2%) THEN 2000
2260 PRINT "2 joy="; joy%(2); ",key="; key%(2); ",left="; l2%; ",right="; r2%
2590 GOTO 2000
8000 REM      initialization
8010 HOME: TEXT: true% = 1
8020 sign% = ASC("#"): asterisk% = ASC("*")
8400 RETURN

```

FIGURE 8-3 READPDL Program.

Reading Controller Values

The *READPDL* program reads the controller values by using the PDL function. The PDL function returns 14 different values depending on value or argument inside the parentheses. The seven odd numbers (1,3,5,7,9,11,13) return values for controller one. The seven even

numbers (0,2,4,6,8,10,12) return values for controller two. Figure 8-4 summarizes the type of information about the controller you can get by using the PDL function. See PDL in Appendix B for a more thorough explanation and a program that prints out all nonzero values of all the PDL values.

<i>Function</i>	<i>Controller</i>	<i>Type Data</i>	<i>Value returned</i>
PDL(0)	2	Paddle wheel	Moving joystick up returns lower values until 0. Moving joystick down returns higher values until 255.
PDL(1)	1		
PDL(2)	2	Paddle wheel	Moving joystick left returns lower values until 0. Moving joystick right returns higher values until 255.
PDL(3)	1		
PDL(4)	2	Joystick	Indicates direction joystick moved by player 0 = Straight up (not pushed) 1 = Pushed up 2 = Pushed right 3 = Pushed up to right 4 = Pushed down 6 = Pushed down to right 8 = Pushed left 9 = Pushed up to left 12 = Pushed down to left
PDL(5)	1		
PDL(6)	2	Left button	0 = not pressed
PDL(7)	1	Yellow*	1 = pressed
PDL(8)	2	Right button	0 = not pressed
PDL(9)	1	Red*	1 = pressed
PDL(10)	2	Keypad	ASCII value of number or symbol pressed. Keypad number 0-9 returns value 48-57. # returns value 35; * returns value 42. No key pressed returns value 0.
PDL(11)	1		
PDL(12)	2	Keypad	Binary value of number. Keypad number 0-9 returns 0-9. # returns value 11; * returns value 10. No key pressed returns value 15.
PDL(13)	1		

* Super controller button color.

FIGURE 8-4 Paddle Functions.

Unlike the INPUT and GET commands, which wait for the operator to respond, the PDL function doesn't wait but just returns the present value of the status of the controller. A zero value means that the operator didn't use that controller function.

The LETTRCHASE program uses the controller as follows:

Keypad	# ends the game. * restarts the game.
Buttons	The player presses either the left or right button to allow the token symbol to move at the start of the game or after matching a letter in the target word.
Joystick	To determine which of the eight directions to move the player token.

Forest of Letters

Figure 8-1 showed that LETTRCHASE displays 21 rows of random letters in columns 2 to 30. The game does not use columns 1 and 31 because many television sets, in generating the image on the screen, overscan or overflow the screen's left and right edges so you can't see all of these columns. If you are using a video monitor without the overscan problem, you could modify the program to use columns 1 and 31.

BASIC has no command to read back a character already displayed on the screen. So the game can check if the letter under the player's token matches the next letter in the target word, a copy of the letter forest must be kept in memory. The two-dimensional integer array *letter%* stores the integer value of each ASCII letter display at column and row location on the text screen.

Random Numbers

Letter Chaser, like many games, uses random numbers to create the element of chance. Blackjack/Poker uses random numbers to shuffle the cards into a random order so that each time you play, the cards are dealt in a different order. This game uses random numbers to select a different target word each time you play (the sur-

prise feature) and mixes the letters up in the letter-forest search area.

You use the random-number function to get a decimal fraction in the range from 0 to 1. It returns a number like .4568943, so if you are storing the number assign it to a real variable (which holds decimal fractions), not an integer variable (which holds only whole numbers). Usually you require a random number within a range of numbers. To convert the random number to a value within the range, multiply the random number by the needed range and add the result to the low-range value. If the low-range value is zero you can omit the add step.

For example, this game needs to generate a series of random capital letters. Capital A has the ASCII value 65 and Z has a value 90, giving a range of 26 values ($90 - 65 + 1$). To get a random capital letter from A through Z, you multiply the needed range (26) by the value returned by the random number function and add the integer result to the low-range value, 65—the ASCII value of A.

Every time you see "You type," this is a command for you to type on ADAM after you have loaded SmartBASIC into memory. Follow all the instructions exactly as the *Companion* explains.

You type `r = RND(5)`

You type `PRINT "Letter=";CHR$(65+INT(26*r));"
RND(5)=";r`

ADAM displays `Letter=T,RND(5)=.732004777`

You should get the same result because BASIC always starts with the same series of random numbers. As you write LETTRCHASE you will learn how to "seed" a random number so you get a different series of random numbers to create true unpredictability.

LET'S WRITE THE PROGRAM

Make sure you have the READPDL program LOAded, because the statements in Figure 8-5 modify it to make it the beginnings of LETTRCHASE. Enter the following statements, which display the forest on the screen.


```
200 DIM letter%(31, 21)
8150 GOSUB 8000: REM letter forest
8500 REM random letters
8510 TEXT: HOME: INVERSE
8530 FOR row = 1 TO lastrow%
8540 NORMAL: PRINT " "; : INVERSE
8550 FOR column = 2 TO 30
8560 letter% = 65+INT(26*RND(5))
8565 letter%(column, row) = letter%
8570 PRINT CHR$(letter%);
8580 NEXT column
8585 NORMAL: PRINT " "; : INVERSE
8590 NEXT row
8990 RETURN
```

FIGURE 8-5 Letter Forest.

Change the name in the line number 10 REM statement to LETTRCHASE and SAVE the program.

You type SAVE lettrchase

After ADAM saves the program, RUN the program.

You will see a forest of random letters appear on the screen. The INVERSE command causes all subsequent text screen characters to be displayed black against a white (on our TV a light tan) background. For LETTRCHASE it shows the player the boundaries of the forest. The NORMAL command changes subsequent text display back to white (multicolored) characters on a black background. It is used here to print a black square at the beginning and end of each text line (columns 1 and 31). These are not in the forest.

Seeding a Random Number

If you RUN the program a few times you notice that the same letters always appear at the same locations. This is because BASIC always starts the random-number sequence with the same starting number. While the numbers in the series are random (in no particular order), the sequence is predictable (always the same for each starting, or seed, number). To get a different random series each time you run the game, you need a random event to allow you to generate a random starting number. How do we do this? The player pushing a number on the keypad to select the one- or two-player game option is our random event. While waiting for the player to push the keypad, the program loops, subtracting 1 from a decimal variable named *seed*.

The program subtracts because the RND function requires a negative number to start a different series of random numbers. The computer subtracts very quickly, and the player reacts to the option prompt screen at different speeds each time, so the value of *seed* at the start of each game is truly random and unpredictable.

Enter the statements to display the option screen and calculate *seed*.

```

8140 GOSUB 8300: REM      intro
8290 RETURN
8300 REM      intro
8310 HTAB 8: VTAB 10: PRINT "LETTER CHASE"
8320 HTAB 8: VTAB 12: PRINT " 1  ONE PLAYER"
8330 HTAB 8: VTAB 14: PRINT " 2  TWO PLAYER"
8335 HTAB 8: VTAB 16: PRINT " PRESS KEYPAD"
8340 onetwo% = PDL(11)
8350 seed = seed-1
8360 IF onetwo% THEN 8400
8370 onetwo% = PDL(10)
8380 IF NOT onetwo% THEN 8340
8400 IF onetwo% = ASC("1") OR onetwo% = ASC("2") THEN RETURN
8410 GOTO 8340

```

FIGURE 8-6 Intro and Seed.

SAVE and RUN the program. Note the new pattern of letters in the forest.

Reading the Words

The LETTERCHASE game uses the string array *words\$* to store the word list from which it selects at random the target word that the players must find letters for in the forest.

Enter the FOR . . . NEXT loop to READ all the possible target words from the DATA statements into the *words\$* string array. Game names were selected for the 10 words.

```

8170 GOSUB 9000: REM      load.word
9000 REM      words
9100 DATA ZAXXON,DONKEY,TURBO,PEPPER,ROCKY
9110 DATA SUBROC,CARNIVAL,COSMIC,SLITHER,LOOPING
9200 words% = 9
9210 FOR word = 0 TO words%
9220 READ words$(word)
9230 NEXT word
9290 RETURN

```

FIGURE 8-7 Load Word.

Check that the words are in the array.

You type For w=1 to 10: ?words\$(w):next

Later you will modify the program to read the words from a file you create with SmartWriter. By putting new words in a file, you can easily change the word list to suit your interest or education level, without changing the BASIC program. Unless they must, programmers never change a working program. It's too easy to make a mistake. If you must change a program, test it afterward to make sure it still works as intended.

Selecting and Displaying the Target Word

The select-word subroutine randomly selects a word from the string array *word\$*, displays it for each player with dashes beneath, and stores the word in the *target%* integer array. Enter the following statements.

```

8030 face$ = CHR$(2): eye$ = CHR$(15)
8050 lastrow% = 21
8070 player$(1) = face$: player$(2) = eye$
8080 track$(1) = " ": track$(2) = "."
8100 beep$ = CHR$(7)+CHR$(7)
8180 GOSUB 9400: REM          get word
9400 REM          select word
9410 word = INT(words%*RND(5))
9420 word$ = words$(word)
9430 NORMAL
9440 VTAB 23: HTAB 2: PRINT face$: TAB(4): word$:
9445 PRINT TAB(16): eye$: TAB(19): word$
9450 dc$ = "-": FOR i = 1 TO LEN(word$): dash$ = dash$+dc$: NEXT i
9460 PRINT TAB(4): dash$: TAB(19): dash$:
9470 column%(1) = 2: column%(2) = 30
9480 FOR player = 1 TO 2
9485 row%(player) = 1: wp%(player) = 1
9487 HTAB column%(player): VTAB row%(player): PRINT player$(player):
9490 FOR wp = 1 TO LEN(word$)
9500 target%(player, wp) = ASC(MID$(word$, wp, 1))
9510 NEXT wp
9520 NEXT player
9530 pp%(1) = 3: pp%(2) = 18
9540 winner% = false%
9590 RETURN

```

FIGURE 8-8 Select Word.

By multiplying the number of words in the list by a random number, the select-word routine calculates a subscript to access a word in the string array, *word\$*, where all the possible words have been stored.

On line 23, the subroutine displays the token character and target word for both players, player 1 on the left and player 2 on the right. On line 24, underneath the target word, it displays dashes indicating the letter has not been matched. As the player finds a matching letter,

it replaces the dash with the letter. In this way both players can keep track of their progress.

The player FOR . . . NEXT loop sets each player's starting row to 1. The word position array *WP%* marks each player's progress in finding matching letters for the target word. One is the first position in the word. The column for player 1 is two. Statement 9487 places the face token at its starting position, the upper left-hand corner of the letter forest. The eyes token for player two is placed at the upper right-hand corner in column 30.

The inner FOR . . . NEXT loop stores the selected word into the *target%* integer array using the *MIDS* function to select each letter and the *ASC* function to convert the letter to an integer value.

Moving the Tokens

Now you're going to add a bunch of statements that move the players' tokens. A player can move in eight directions using the joystick controller—up, down, right, left and diagonally toward each of the four corners. A zero value from the joystick means no movement. BASIC treats any nonzero value as true, so if the joystick PDL function returns a value, that player's token is moved. It's possible for both players to move at the same time, so the routine checks if both joysticks returned a value.

SAVE all that work. LIST and check that you entered everything correctly. RUN it and use the controllers to move the tokens. (Figure 8-9, p. 182)

The player-direction subroutine puts a track character at the current location, a blank for player 1 and a dot for player 2. This way, they both can see where they have been.

To make the ON . . . GOSUB multiway switch work correctly, the value of a northwest movement is changed from 12 to 7. Depending on the value of the joystick direction value, the GOSUB transfers execution to the proper movement subroutine. Each movement subroutine returns to the statement following the ON . . . GOSUB. Then the program displays the player's tokens at the new column and row position.

The movement routines change the value of the player column and/or row value. They allow wraparound movement. For example, the player can go from the top to the bottom line by moving up. If the routines find a column or row value at the limit, the value is just reset to the opposite limit. For example, the *row%* variable has limits 1 (the


```

2300 IF joy1(1)+joy1(2) THEN GOSUB 4000
4000 REM          joystick direction
4020 IF joy1(1) THEN player1 = 1: GOSUB 4100
4030 IF joy1(2) THEN player1 = 2: GOSUB 4100
4090 RETURN
4100 REM          process joystick direction
4110 INVERSE
4120 seed = seed-1: REM   for next game
4150 HTAB column1(player1): VTAB row1(player1): PRINT track$(player1):
4210 IF joy1(player1) = 12 THEN joy1(player1) = 7
4220 direct1 = joy1(player1)
4230 ON direct1 GOSUB 5100, 5200, 5300, 5400, 5500, 5600, 5700, 5800, 5900
4300 REM          display player graphic symbol
4320 HTAB column1(player1): VTAB row1(player1): PRINT player$(player1):
4699 RETURN
5100 REM   north(up)
5120 IF row1(player1) <= 1 THEN row1(player1) = lastrow1+1
5130 row1(player1) = row1(player1)-1: RETURN
5190 RETURN
5200 REM   east(right)
5220 IF column1(player1) >= 30 THEN column1(player1) = 1
5230 column1(player1) = column1(player1)+1
5290 RETURN
5300 REM   NE(up,right)
5320 GOSUB 5100: REM   row up
5340 GOSUB 5200: REM   column right
5390 RETURN
5400 REM   south(down)
5420 IF row1(player1) >= lastrow1 THEN row1(player1) = 0
5430 row1(player1) = row1(player1)+1
5490 RETURN
5500 REM   dummy
5590 RETURN
5600 REM   SE(down,right)
5620 GOSUB 5400: REM   row down
5640 GOSUB 5200: REM   column right
5690 RETURN
5700 REM   SW(down,left)
5720 GOSUB 5400: REM   row down
5740 GOSUB 5800: REM   column left
5790 RETURN
5800 REM   west(left)
5820 IF column1(player1) <= 2 THEN column1(player1) = 31
5830 column1(player1) = column1(player1)-1
5890 RETURN
5900 REM   NW(up,left)
5920 GOSUB 5100: REM   row up
5940 GOSUB 5800: REM   column left
5990 RETURN

```

FIGURE 8-9 Moving Tokens.

top row) and 21 (the bottom row of the letter forest). If *row%* equals 1 and the joystick returns an up direction value, setting *row%* to 21 (the opposite limit) moves the token from the top row to the bottom row.

Matching

Matching statements match the letter under the player's token to the next unmatched letter in the target word. The ASCII value of the letter is retrieved from the letter array using the player's column and row that placed the token on the screen. Each element in the array matches a column, row location on the video screen.

On an equal match the routine pops the letter into the dash. If the player matched the full word, the *winner%* variable gets set to *true%*,

otherwise it adds 1 to the player's word position (*WP%*) in the target word. Setting the *pause%* switch false requires the player to press a controller button before moving the token again.

Accessing an element in an array takes more time than getting a value from a simple variable. BASIC must calculate an address from the subscripts. Since the ASCII value of the letter is used twice, first in the test and second to display the letter, it's more efficient to store the integer value of the letter in a variable than to retrieve it twice from the *letter%* array.

```

2240 IF l1%+l2%+r1%+r2% THEN GOSUB 6000: REM button pressed
4200 IF move$(player%) = false$ THEN PRINT beep$: : GOTO 4320
4500 REM check for matching letter
4510 letter% = letter$(column$(player%), row$(player%))
4530 IF letter% = target$(player%, wp$(player%)) THEN 4630
4540 PRINT CHR$(7): : REM one beep
4590 RETURN
4600 INVERSE
4610 HTAB pp$(player%)+wp$(player%): VTAB 24: PRINT CHR$(letter%):
4630 IF wp$(player%) = LEN(word$) THEN winner% = true$: RETURN
4640 wp$(player%) = wp$(player%)+1
4650 move$(player%) = false$
6000 REM button pressed
6020 IF l1%+r1% THEN move$(1) = true$
6040 IF l2%+r2% THEN move$(2) = true$
6090 RETURN

```

FIGURE 8-10 Match.

The Winner

Enter the statements announcing the winner to the main routine. After announcing the winner, it asks if you want to play again. Either player presses number sign (#) to end or asterisk (*) to play again.

```

2590 IF NOT winner% THEN 2000
2600 REM winner
2630 PRINT TAB(10); "PLAYER "; player%; " WINS!!!"
2640 PRINT "PLAY AGAIN?(*=Yes/#=No)";
2650 key$(1) = PDL(11): key$(2) = PDL(10)
2660 IF key$(1) = sign% OR key$(2) = sign% THEN TEXT: END
2670 IF key$(1) = asterisk% OR key$(2) = asterisk% THEN 2700
2680 GOTO 2650
2700 GOSUB 8500: REM letters
2710 GOSUB 9400: REM new word
2990 GOTO 2000: REM play again

```

FIGURE 8-11 Winner.

To play again, the routine GOSUBs and starts the letter-forest creation routine and the word-selection routine again.

Kaleidoscope

A clever teenage computer programmer, Ed Milkow, wrote the Kaleidoscope routine, which dazzles the eyes with lines of rapidly changing colors, a reward for the winner. We leave unexplained how it works, as a challenge for you to figure out. (Hint: the variable named *row* is used both as a column and a row.)

```
2620 GOSUB 7000: REM  colors
7000 REM  WIN ROUTINE
7030 GR
7040 FOR row = 0 TO 39
7050 HLIN row, 39-row AT row: VLIN row, 39-row AT row
7055 HLIN row, 39-row AT 39-row: VLIN row, 39-row AT 39-row
7070 hue = hue+1: IF hue = 16 THEN hue = 1
7080 COLOR = hue
7090 NEXT
7099 RETURN
```

FIGURE 8-12 Kaleidoscope.

Making the Computer Be Player 2

You turn a competitive two-player game, like Letter Chaser, into a one-player game by making the computer the second player. In Letter Chaser, each player's actions generate two inputs—a direction movement value from the joystick and a GO command at the start and after each letter match by pushing either of the controller buttons. The computer-player routine always sets the *pause%* switch to false so the computer never pauses after a match. A random-number calculation provides a value for the direction movement. A DELAY loop prevents the computer from moving too fast.

```
2260 IF onetwo = ASC("1") THEN GOSUB 3000: REM  computer
3000 REM  computer player two
3010 IF pause% THEN pause% = pause%-1: RETURN
3020 pause% = delay%
3030 joy% (2) = 9*RND(5): REM  9 directions including dummy
3040 move% (2) = true%
3190 RETURN
8090 delay% = 10
```

FIGURE 8-13 Computer Player Two.

To give the computer some intelligence, add a scan routine that looks ahead in all directions to find the matching letter. Instead of a random direction, make the direction of the letter the joystick movement value. Add a difficulty level to the game by increasing the range of the scan so the computer sees more letters before deciding which way to move.

The Final Listing

Figure 8-14 shows the final listing. If your program doesn't work, do a line-by-line match. Look for missing semicolons, statements incorrectly numbered and misspelled variable names.

```

10 REM  letterchase 01/21/84
100 DIM joy%(2), key%(2), move%(2), row%(2), column%(2)
200 DIM letter$(31, 21), words$(20), target$(2, 12)
1000 REM      main game process
1020 GOSUB 8000: REM      initialization
2000 REM      read controllers
2020 joy%(1) = PDL(5): REM      joystick 1
2040 joy%(2) = PDL(4): REM      joystick 2
2060 key%(1) = PDL(11): REM      keypad 1
2080 key%(2) = PDL(10): REM      keypad 2
2100 IF key%(1) = sign% OR key%(2) = sign% THEN TEXT: END
2120 IF key%(1) = asterisk% OR key%(2) = asterisk% THEN 1030
2200 l1% = PDL(7): REM      left button 1
2210 l2% = PDL(6): REM      left button 2
2220 r1% = PDL(9): REM      right button 1
2230 r2% = PDL(8): REM      right button 2
2240 IF l1%+l2%+r1%+r2% THEN GOSUB 6000: REM      button pressed
2260 IF onetwo% = ASC("1") THEN GOSUB 3000
2300 IF joy%(1)+joy%(2) THEN GOSUB 4000
2590 IF NOT winner% THEN 2000
2600 REM      winner
2620 GOSUB 7000: REM      colors
2630 PRINT TAB(10): "PLAYER "; player%; " WINS!!!"
2640 PRINT "PLAY AGAIN?(*-Yes/*-No)":
2650 key%(1) = PDL(11): key%(2) = PDL(10)
2660 IF key%(1) = sign% OR key%(2) = sign% THEN TEXT: END
2670 IF key%(1) = asterisk% OR key%(2) = asterisk% THEN 2700
2680 GOTO 2650
2700 GOSUB 8500: REM      letters
2710 GOSUB 9400: REM      new word
2990 GOTO 2000: REM      play again
3000 REM      computer player two
3010 IF pause% THEN pause% = pause%-1: RETURN
3020 pause% = delay%
3030 joy%(2) = 9*RND(5)
3040 move%(2) = true%
3190 RETURN
4000 REM      joystick direction
4020 IF joy%(1) THEN player% = 1: GOSUB 4100
4030 IF joy%(2) THEN player% = 2: GOSUB 4100
4090 RETURN
4100 REM      process joystick direction
4110 INVERSE
4120 seed = seed-1
4150 HTAB column%(player%): VTAB row%(player%): PRINT track$(player%);
4200 IF move%(player%) = false% THEN PRINT beep$: GOTO 4320
4210 IF joy%(player%) = 12 THEN joy%(player%) = 7
4220 direct% = joy%(player%)
4230 ON direct% GOSUB 5100, 5200, 5300, 5400, 5500, 5600, 5700, 5800, 5900
4300 REM      display player graphic symbol
4320 HTAB column%(player%): VTAB row%(player%): PRINT player$(player%);
4500 REM      check for matching letter
4510 letter% = letter$(column%(player%), row%(player%))
4530 IF letter% = target%(player%, wp%(player%)) THEN 4600
4540 PRINT CHR$(7): REM      one beep
4590 RETURN
4600 INVERSE
4610 HTAB wp%(player%)+wp%(player%): VTAB 24: PRINT CHR$(letter%);
4630 IF wp%(player%) = LEN(words%) THEN winner% = true%: RETURN
4640 wp%(player%) = wp%(player%)+1
4650 move%(player%) = false%
4690 RETURN
5100 REM      north(up)
5120 IF row%(player%) <= 1 THEN row%(player%) = lastrow%+1
5130 row%(player%) = row%(player%)-1
5190 RETURN
5200 REM      east(right)
5220 IF column%(player%) >= 30 THEN column%(player%) = 1
5230 column%(player%) = column%(player%)+1
5290 RETURN
5300 REM      NE(up,right)
5320 GOSUB 5100: REM      row up

```



```

5340 GOSUB 5200: REM          column right
5390 RETURN
5400 REM          south(down)
5420 IF row$(player) >= lastrow$ THEN row$(player) = 0
5430 row$(player) = row$(player)+1
5490 RETURN
5500 REM          dummy
5590 RETURN
5600 REM          SE(down,right)
5620 GOSUB 5400: REM          row down
5640 GOSUB 5200: REM          column right
5690 RETURN
5700 REM          SW(down,left)
5720 GOSUB 5400: REM          row down
5740 GOSUB 5800: REM          column left
5790 RETURN
5800 REM          west(left)
5820 IF column$(player) <= 2 THEN column$(player) = 31
5830 column$(player) = column$(player)-1
5890 RETURN
5900 REM          NW(up,left)
5920 GOSUB 5100: REM          row up
5940 GOSUB 5800: REM          column left
5990 RETURN
6000 REM          button pressed
6020 IF l1+r1 THEN move$(1) = true$
6040 IF l2+r2 THEN move$(2) = true$
6090 RETURN
7000 REM          WIN ROUTINE
7030 GR
7040 FOR row = 0 TO 39
7050 HLINE row, 39-row AT row: VLINE row, 39-row AT row
7055 HLINE row, 39-row AT 39-row: VLINE row, 39-row AT 39-row
7070 hue = hue+1: IF hue = 16 THEN hue = 1
7080 COLOR = hue
7090 NEXT
7099 RETURN
8000 REM          initialization
8010 TEXT: HOME: true$ = 1
8020 sign$ = ASC("#"): asterisk$ = ASC("*")
8030 face$ = CHR$(2): eyes$ = CHR$(15)
8050 lastrow$ = 21
8070 player$(1) = face$: player$(2) = eyes$
8080 track$(1) = " ": track$(2) = "."
8090 delay$ = 10
8100 beep$ = CHR$(7)+CHR$(7)
8140 GOSUB 8300: REM          intro
8150 GOSUB 8500: REM          screen
8170 GOSUB 9000: REM          load word
8180 GOSUB 9400: REM          get word
8290 RETURN
8300 REM          intro
8310 HTAB 8: VTAB 10: PRINT "LETTER CHASE"
8320 HTAB 8: VTAB 12: PRINT " 1 ONE PLAYER"
8330 HTAB 8: VTAB 14: PRINT " 2 TWO PLAYER"
8335 HTAB 8: VTAB 16: PRINT " PRESS KEYPAD"
8340 onetwo$ = PDL(11)
8350 seed = seed-1
8360 IF onetwo$ THEN 8400
8370 onetwo$ = PDL(10)
8380 IF NOT onetwo$ THEN 8340
8400 IF onetwo$ = ASC("1") OR onetwo$ = ASC("2") THEN RETURN
8410 GOTO 8340
8490 RETURN
8500 REM          random letters
8510 TEXT: HOME: INVERSE
8520 random = RND(seed)
8530 FOR row = 1 TO lastrow$
8540 NORMAL: PRINT " "; : INVERSE
8550 FOR column = 2 TO 30
8560 letter$ = 65+INT(26*RND(5))
8565 letter$(column, row) = letter$
8570 PRINT CHR$(letter$);
8580 NEXT column
8585 NORMAL: PRINT " "; : INVERSE
8590 NEXT row
8990 RETURN
9000 REM          words
9100 DATA ZAXXON,DONKEY,TURBO,PEPPER,ROCKY

```



```

9110 DATA SUBROC,CARNIVAL,COSMIC,SLITHER,LOOPING
9200 words% = 9
9210 FOR word = 0 TO words%
9220 READ words$(word)
9230 NEXT word
9290 RETURN
9400 REM select word
9410 word = INT(words%*RND(5))
9420 words% = words$(word)
9430 NORMAL
9440 VTAB 23: HTAB 2: PRINT face$: TAB(4): words%
9445 PRINT TAB(16): eyes$: TAB(19): words%
9450 dc$ = "-": dash$ = ""
9455 FOR i = 1 TO LEN(words%): dash$ = dash$+dc$: NEXT i
9460 PRINT TAB(4): dash$: TAB(19): dash%
9470 column%(1) = 2: column%(2) = 30
9480 FOR player = 1 TO 2
9485 row%(player) = 1: wp%(player) = 1
9487 HTAB column%(player): VTAB row%(player): PRINT player$(player)
9490 FOR wp = 1 TO LEN(words%)
9500 target%(player, wp) = ASC(MID$(word$, wp, 1))
9510 NEXT wp
9520 NEXT player
9530 pp%(1) = 3: pp%(2) = 18
9540 winner% = false%
9590 RETURN

```

FIGURE 8-14 LETTRCHASE Final Listing.

Reading Words from a File

Before adding the following statements that replace the DATA statements and READ statements, you may want to SAVE the program under a different name, so you have one version that reads the program's own words and another that reads the words from files you create.

```

200 DIM letter%(31, 21), word$(100), target%(2, 12)
6000 REM read word file
6010 d$ = CHR$(4)
6020 word% = 0
6050 PRINT d$; "open "; name$
6100 PRINT d$; "read "; name$
6150 ONERR GOTO 6600
6200 INPUT " "; word$(word%)
6400 word% = word%+1
6500 GOTO 6200
6600 REM ONERR routine
6610 CLRERR
6700 PRINT d$; "close "; name$
6710 IF ERRNUM(0) = 5 AND word% THEN RETURN
6750 IF NOT word% THEN PRINT d$; "delete "; name$: RETURN
6990 RETURN
9020 INPUT "filename="; name$
9050 IF LEN(name$) = 0 THEN 9200
9060 GOSUB 6000: REM read file
9070 IF word% THEN 9400
9080 PRINT "FILE "; name$; " NOT FOUND"
9090 GOTO 9020

```

FIGURE 8-15 Reading From a File.

These statements read the *chasewords* file from the left tape drive. You must create the *chasewords* file as described in the next section. The *word%* variable counts the records read, and is the subscript to store the words in the *word\$* string array. If you have more than 100 words you must change the size of *word\$* DIMension statement to provide more storage space.

Creating a Wordfile

SmartBASIC cannot read a file created by SmartWriter, but SmartWriter can read a file created by SmartBASIC. Use the following program to create *chasewords* as a BASIC file. Then you can use SmartWriter to add, change and delete words from the list. Remember, only one word per line, no spaces in the word, and the word must end with the return triangle.

```
10 REM create file
1000 d$ = CHR$(4)
1100 INPUT "FILENAME="; name$
1200 PRINT d$; "open "; name$
1300 PRINT d$; "write "; name$
1400 PRINT "word"
1500 PRINT d$; "close "; name$
```

FIGURE 8-16 Creating a File.

Enhancements

Allow the players to enter their names, so the game can announce the winner by name.

Have secret holes that magically transport the player to someplace else in the forest.

Allow the player to enter the name of word file so the player can move up to harder words.

Have Fun

Games are fun to play and write. Now invent one yourself! Write games for publication in *ADAM Family Computing* and make some money to pay for all the new equipment you will want to add to your ADAM.

CHAPTER NINE

How to Write Music on ADAM

(Even if You Can't
Read a Single Note)

The thumping rhythm of music pervades our lives. As we travel, it plays on our car radios. The laughter, drama and adventure of our movies and television wouldn't be the same without the melody and rhythm of music. Video games without sounds and music would not be as enjoyably exciting to play or watch. You can attribute the popularity of Coleco games not only to their exceptional playability and superb color graphics, but also to the charming melodies and sounds that accompany the action. Without music, *Carnival* and *Mousetrap* would lose their wonderful appeal.

MAKING MUSIC

Have you ever blown notes through a horn, run a bow over the strings of a violin making them sing, or struck the ivory keys of a piano filling the air with melody? No? Unfortunately, ADAM cannot give you those wonderful unique experiences. It takes years of practice to learn how to play an instrument. But the *Companion* teaches you how to program ADAM to play music. You'll learn how to read sheet music so you can incorporate the musical notes into your own BASIC programs and games.

In Chapter 7 you wrote the *PICMAKER* program. This chapter presents a similar program called *MUSICMAKER*, which allows you to enter a melody instead of a picture. A melody consists of notes

organized by the composer into a series of agreeable sounds, such as you hear in a popular song. You'll use a simple note code to enter the melody notes. The MUSICMAKER program, in addition to playing the melody through your television speaker, computes the numeric values to incorporate into your BASIC program to make ADAM play the tune. If you want to experiment, you may use the MUSICMAKER program to compose music.

SOUNDS

Vibrations produce the sensation of sound you hear and feel. Place your hand over your radio's speaker and you feel the air vibrating. Radio signals are turned into electrical pulses causing a thin sheet inside the speaker to move back and forth, vibrating, pushing the air against your hand. Like the radio, ADAM can produce vibrations. It has a computer chip that you control by sending it numbers, rather than radio signals, to create the sound vibrations produced by your television speaker.

Frequency, Duration and Loudness

By playing certain ear-pleasing frequencies for different short durations at various loudnesses, musicians—and speakers—produce music.

Frequency, which musicians call *pitch*, defines how fast the speaker vibrates over time, usually measured in seconds. One back-and-forth movement of the sheet inside the speaker is a cycle. If the speaker vibrates 20 cycles a second, it has a frequency of 20 Hertz (Hz). The higher the frequency or pitch, the sharper the sound. Low-frequency vibrations create a deep bass sound like a foghorn. Figure 9-1 shows the frequency values in Hertz for the range of musical notes that you can play on ADAM. Traditionally, composers have used the seven letters A through G to identify note pitches within an octave. Octaves divide the entire range of musical frequencies into proportional groups. The pitches in the higher octave have a frequency value twice those in the lower octave. For example, the note A in the fourth octave has a pitch of 440, which is twice the 220 value of the A note in the third octave. This proportionality, which exists throughout music, is what makes music sound pleasant to us.

<i>Note</i>	<i>Octave</i>	<i>Frequency</i>	<i>Note</i>	<i>Octave</i>	<i>Frequency</i>
C	2	65.405	C	3	130.81
C#	2	69.295	C#	3	138.59
D	2	73.415	D	3	146.83
D#	2	77.783	D#	3	155.57
E	2	82.408	E	3	164.82
F	2	87.308	F	3	174.62
F#	2	92.498	F#	3	185.
G	2	97.998	G	3	196.
G#	2	103.83	G#	3	207.65
A	2	110.	A	3	220.
A#	2	116.64	A#	3	233.08
B	2	123.47	B	3	246.94
C	4	261.62	C	5	523.24
C#	4	277.18	C#	5	554.36
D	4	293.66	D	5	587.32
D#	4	311.13	D#	5	622.26
E	4	329.63	E	5	659.26
F	4	349.23	F	5	698.46
F#	4	370.	F#	5	740.
G	4	392.	G	5	784.
G#	4	415.3	G#	5	830.6
A	4	440.	A	5	880.
A#	4	466.16	A#	5	932.32
B	4	493.88	B	5	987.76
C	6	1046.5	C	7	2093.
C#	6	1108.72	C#	7	2217.44
D	6	1174.64	D	7	2349.38
D#	6	1244.52	D#	7	2485.04
E	6	1318.52	E	7	2637.04
F	6	1396.92	F	7	2793.84
F#	6	1480.	F#	7	2960.
G	6	1568.	G	7	3136.
G#	6	1661.2	G#	7	3322.4
A	6	1760.	A	7	3520.4
A#	6	1864.64	A#	7	3729.28
B	6	1927.52	B	7	3855.04

FIGURE 9-1 Frequency Values for Musical Notes.

Music divides an octave into 12 musically equal parts called half steps; combining two half steps produces a whole step. To identify the 12 pitches in an octave, Figure 9-1 uses the letter A through G either alone or with the sharp symbol (#). The letter of the lower pitch and

the sharp symbol (#) identify the frequency of the higher step. The number after the half-step identification defines the octave. For example, on the piano keyboard C4 means note C in the fourth octave, which has a pitch of 261.50 Hz. The sharp symbol in C#4 moves the C4 pitch up a half step to 277.77 Hz. Notice that from E to F and from B to C is a half step. Sharps are rarely written for E or B because E# is F and B# is C in the next-higher octave. The flat symbol (b) sends the letter pitch down a half step. The pitch Db is the same as the pitch C#. Since you can identify all pitch values using the sharp symbol, for programming simplicity a MUSICMAKER only accepts the sharp symbol.

As music plays and as we speak, the frequency of sound changes. *Duration* measures how long a sound stays at a certain frequency. Musical durations, shown in Figure 9-2, are proportional and relative. Beginning with the whole note, each shorter note duration has a relative time-period value proportionally half the preceding one. For example, in MUSICMAKER we assign the whole note a time value of 64. The half note has a value half of the whole note, or 32 (64/2). Because you can play music fast or slow, the note-duration values are relative. A tempo value defines how much faster to play all the notes. To keep the melody the same, if you play the whole notes twice as slow, you must play the half notes twice as slow. In MUSICMAKER the *tempo* value is multiplied by the relative duration of the note to determine the amount of time to sound each pitch.

Turning up the volume on your radio increases the strength or *loudness* of the sound. Striking a piano key harder increases the loudness, but you still hear the same pitch. The air pulses with greater intensity but still at the same frequency. In MUSICMAKER all notes are played with the same loudness.

HOW ADAM MAKES SOUNDS

ADAM uses a Texas Instruments SN76489A sound chip to create the musical and explosion sounds you hear in Coleco's games. The chip has three frequency-tone channels and a noise channel. Noise consists of a mixture of many different frequencies, similar to white light, which consists of a mixture of light in all different colors. The program example for the CALL command in Appendix B illustrates how to create the noise of an explosion.












Note symbol	Letter code	Relative duration value	Description
	W	64	Whole note
	H:	54	Double dotted half note
	H.	48	Dotted half note
	H	32	Half note
	Q:	28	Double dotted quarter
	Q.	24	Dotted quarter
	Q	16	Quarter note
	E.	12	Dotted eighth note
	E	8	Eighth note
	S	4	Sixteenth note
	T	2	Thirty-second note

FIGURE 9-2 Commonly Used Duration Symbols and Values.

Tone refers to the sound made by a pitch or frequency. Making sounds with the three tone channels is like striking three piano keys at the same time: you get a richer sound. However, you need only one finger to play a melody on a piano, so the *Companion* will teach you to use one tone channel to play a melody on ADAM. With the technical information about the sound chip provided below, you can experiment using the three channels and make noises of your own.

By sending an integer value to the chip, you can set the pitch or frequency of the sound and the volume intensity or loudness of the sound. This is similar to using the CHR\$ function to display a letter on the screen, but instead of a letter or number symbol on the screen, the numeric value produces a sound. You make ADAM play a musical note by sending the chip a value to define a musical frequency and then, after a certain duration, you change the frequency to play another note.

To send the number to the chip you need a very small Z-80 microprocessor program. The following BASIC program uses such a program to send numbers to the chip to define the sound's volume and frequency. For simplicity, the program sets the loudness to the highest level. If you find it too loud, turn down the volume on your television. The program POKEs a 6-byte Z-80 program into memory beginning at 28000. The Z-80 program loads microprocessor register A with the byte you POKE into 28006, outputs it to port 255, and returns to BASIC. To make the program more understandable, the variable *chip%* stores the address where you POKE the decimal value you send to the TI sound chip. The variable *sound%* stores the start address of the Z-80 program used by the CALL command. The CALL *sound%* executes the microprocessor program that sends the number to the sound chip.

Type and SAVE the following SOUND program (Figure 9-3). It will evolve into the MUSICMAKER program.

Run the SOUND program and enter a few of the pitches shown in Figure 9-1 for the relative duration values shown in Figure 9-2. Listen to the sound produced. Was it pleasing to your ear? The SOUND program lets you play one note, but a later version will store a series of notes in memory so ADAM actually plays a music piece.


```

5 LOMEM :29000
10 REM      sound 1/21/84
900 GOSUB 16700: true% = 1
1000 REM    SOUND MAIN LOOP
1610 GOSUB 2800: REM      INPUT
1630 IF exit% THEN END
1990 GOTO 1000
2800 REM    INPUT PITCH
2810 INPUT "pitch="; pitch$
2820 IF LEN(pitch$) = 0 THEN exit% = true%: RETURN
2830 pitch% = VAL(pitch$)
2840 INPUT "duration="; duration$
2850 duration = VAL(duration$)
2860 GOSUB 6000: REM      MAKE SOUND
2990 RETURN
6000 REM    PLAY NOTE OR SOUND
6005 pitch% = mhertz/(32*pitch%)
6010 second% = pitch%/16
6020 first% = 128+register%+(pitch%-second%*16)
6030 POKE chip%, first%: CALL sound%
6040 POKE chip%, second%: CALL sound%
6050 POKE chip%, loud%: CALL sound%
6100 FOR delay = 1 TO duration*tempo: NEXT delay
6120 POKE chip%, quiet%: CALL sound%
6300 RETURN
16700 REM    INITIALIZE SOUND VALUES
16710 DATA 58,102,109,211,255,201
16720 loud% = 144: quiet% = 159
16730 sound% = 28000
16740 chip% = 28006
16750 mhertz = 3597000
16760 tempo = 20
16800 REM    LOAD Z-80 MICROPROCESSOR PROGRAM
16810 FOR address = sound% TO sound%+5
16820 READ byte%
16830 POKE address, byte%
16840 NEXT address
16990 RETURN

```

FIGURE 9-3 Sound.

Technical Details

Each of the three-tone generator channels contains a 10-bit counter, called a register, which the chip decrements to determine when to send out a vibration. The lower the value you put in the counter, the more frequently the chip sends out a vibration. This produces higher pitch and sharper sound. The chip requires a frequency value

different from the musical pitch values. To compute the chip value, you use the value of ADAM's 3.597 megahertz (3,597,000 cycles per second) microprocessor clock signal in the following formula:

$$\text{chip value} = \text{clock signal frequency} / (32) * \text{musical frequency}$$

For example, the note A in the second octave has a frequency of 110 Hz. Substituting in the formula gives

$$\text{chip value} = 3,597,000 / (32 * 110)$$

$$\text{chip value} = 1,021.875$$

Since the Z-80 processor can only send eight bits at a time to the chip, two numbers are needed to define the frequency. The first number selects the channel and supplies the four low-order or rightmost bits of the 10-bit frequency. The second number supplies the six high-order bits. A high-order bit of zero identifies it to the chip as the second number. Dividing the *pitch%* by 16 shifts out the lower four bits giving the second number. The first number is the sum of three numbers:

1. The number 128, which sets the leftmost bit to 1, identifying the number to the chip as the first or only number.
2. The control register number of the tone channel. Figure 9-4 shows the numbers to address each of the eight-chip control registers.

<i>Decimal Value</i>	<i>Control register</i>
0	Tone channel 1 frequency
16	Tone channel 1 loudness
32	Tone channel 2 frequency
48	Tone channel 2 loudness
64	Tone channel 3 frequency
80	Tone channel 3 loudness
96	Noise sound
112	Noise loudness

To select the register to which you want to send a frequency or loudness value, set the leftmost bit to 1 and add the decimal value of the control register to the loudness value. For example, to turn tone channel 2 to the maximum, add

128	Set high-order bit to 1 indicating first byte
48	Tone channel 2 loudness
0	Maximum loudness from Figure 9-6
176	Value to send to chip

FIGURE 9-4 TI Sound Chip Control Register Values.

Here, the variable *register%* has the value zero addressing the first tone's pitch register. By changing the value of *register%* to 32 or 64 you can set pitches of the two other channels.

3. Multiplying the second number by 16 and subtracting the product from *pitch%* gives the four lower bits for the first number.

Figure 9-5 illustrates the format of the two numbers and shows an example for the pitch 261.62 Hz, the frequency value for middle C.

Associated with each channel is a loudness-control register. To produce sound, after putting a frequency value into the frequency register, you put a number into the loudness register to turn up the volume. The number not only defines loudness but also selects either one of the three music channels or the noise channel. Figure 9-4 defines the value you must add to select the channel and register. Figure 9-6 defines the values for loudness. For example,

To illustrate the format of the two bytes sent to the sound chip, consider the pitch 261.62. To make the sound chip create a sound at this frequency you need to send the value $3,597,000/(32 \times 261.62)$ or 430. In binary 430 appears as:

Position id	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9
Position value	512	256	128	64	32	16	8	4	2	1
Binary value	0	1	1	0	1	0	1	1	1	0

Dividing 430 by 16 gives 26, which you place in positions f0 through f5 in the second byte. The remainder ($430 - 26 \times 16$) of 14, which is 1110 in binary, you place in f6 through f9 in the first byte. To identify it to the sound chip as the first byte, you add 128 which sets the leftmost bit to 1. The register value from Figure 9-4 for channel 1 is zero.

The bytes sent are as follows:

	First byte								Second byte							
	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
id	1	r	r	r	f6	f7	f8	f9	0	x	f0	f1	f2	f3	f4	f5
binary	1	0	0	0	1	1	1	0	0	0	0	1	1	0	1	0
decimal	28 + 0 + 14 = 142								26							

r r r = register number value from Figure 9-4

x = unused

FIGURE 9-5 Format of Frequency Numbers.

<i>Decimal value</i>	<i>Loudness adjustment in decibels</i>
0	maximum volume
1	-2
2	-4
3	-6
4	-8
5	-10
6	-12
7	-14
8	-16
9	-18
10	-20
11	-22
12	-24
13	-26
14	-28 (barely audible)
15	-no volume

FIGURE 9-6 TI Sound Chip Loudness Values.

<i>Decimal value</i>	<i>Type noise</i>
0	periodic noise
4	continuous noise
	<i>Noise shift rate</i>
0	n/512
1	n/1024
2	n/2048
3	voice 3 frequency

n is the 4-megahertz (4,000,000) clock rate used by ADAM.

To set the type noise and shift rate, add the two values with the noise-control register value (96) and the single byte transfer value (128). For example, if you want continuous noise, with the sound varied by the voice 3 frequency, you add:

128	single byte transfer value
96	noise-control register from Figure 9-4
4	continuous noise
3	sound varied by voice 3 frequency
<u>231</u>	value sent to chip

Remember, you won't hear any noise unless you turn the volume on by sending 240 (112 + 128) to the chip.

FIGURE 9-7 Noise-Control Values.

to turn off the noise sound, add the following three numbers together:

128 to set higher-order bit on, indicating first or only number

112 to select the noise-loudness register

15 to turn off the sound

255 value POKEed into location in chip%, 28006

Since the variable *loud%* has the value zero, statement 6050 sets the loudness for the voice channel to the maximum volume.

Figure 9-7 describes how to define characteristics of the noise you want the sound chip to create.

HAPPY BIRTHDAY

It doesn't matter if you don't completely understand the technical explanation; you can still use the program to play music. Make sure you have LOAded the sound program into memory, because the next group of statements (Figure 9-8, p. 200) modify sound to play the ever-popular song "Happy Birthday." After you have successfully played the tune, the *Companion* will explain how the DATA statements that define the notes were created from the symbols and lines on sheet music.

If the song doesn't sound right, check that you entered all the DATA statements and that they are in the correct order.

The BIRTHDAY program uses the READ statement to read the duration and pitch values into the two-dimensional *music%* array. The first dimension is the note: using 100 means you can store the values for 100 notes. The second dimension represents the two values for each note—duration and pitch. Rather than use the meaningless numbers 0 and 1, the program assigns them to the more meaningful variable names *time%* for duration and *frequency%* for pitch.

The *play music* subroutine that begins at statement 3000 is a FOR . . . NEXT loop that passes the *pitch%* and *duration%* values for each of the 25 notes needed to play "Happy Birthday" to the sound subroutine from the *music%* array. SAVE the program with the name BIRTHDAY. You will make more changes to it.

CONVERTING MUSICAL NOTATION TO NUMBERS

Music consists of notes, ear-pleasing pitches played for certain durations. To define these values for musicians, composers use a symbolic


```

10 REM    birthday 1/21/84
90 max = 100: max% = max
100 DIM music%(max, 2)
210 time% = 1: frequency% = 2
610 DATA    8,196,eg3
612 DATA    8,196,eg3
614 DATA    16,220,qa3
616 DATA    16,196,qg3
618 DATA    16,261,qc4
620 DATA    32,246,hb3
630 DATA    8,196,eg3
632 DATA    8,196,eg3
634 DATA    16,220,qa3
636 DATA    16,196,qg3
638 DATA    16,294,qd4
640 DATA    32,262,hc4
642 DATA    8,196,eg3
644 DATA    8,196,eg3
646 DATA    16,392,qg4
648 DATA    16,329,qe4
650 DATA    16,262,qc4
652 DATA    16,247,qb3
654 DATA    24,220,q.a3
656 DATA    8,349,ef4
658 DATA    8,349,ef4
660 DATA    16,330,qe4
662 DATA    16,262,qc4
664 DATA    16,294,qd4
666 DATA    48,262,h.c4
800 FOR music = 1 TO 25
810 READ music%(music, time%), music%(music, frequency%)
820 READ note$
830 NEXT music
2800 REM    play
2810 REM
2820 FOR music = 1 TO 25
2830 pitch% = music%(music, frequency%)
2840 REM
2850 duration = music%(music, time%)
2860 GOSUB 6000: REM    MAKE SOUND
2870 NEXT music
2880 INPUT "PLAY AGAIN(Y/N)?"; ans$
2890 IF ans$ = "Y" OR ans$ = "y" THEN RETURN
2900 exit% = true%
2990 RETURN

```

FIGURE 9-8 Changes to Sound to Create BIRTHDAY Program.

notation written on two sets of five lines called *staves*. Figure 9-9 shows the two staves: the top is called the G-clef and the bottom one the bass or F-clef. The letters written on and between the lines represent the

itches of the music notes, and Figure 9-1 defines their values. Each repetition of the seven letters A through G, starting with the letter C, begins an *octave*. Between the two staves is the pitch middle C, which begins the *fourth* octave.

The note symbols shown in Figure 9-2 define the duration of the sound, while the placement of the symbol on the two staves defines the pitch. To play music, ADAM needs numbers for duration and pitch instead of symbols and placement on a staff. Here's how to convert sheet music for "Happy Birthday," shown in Figure 9-10 (next page), to the numbers.

We start with the first note symbol on the left.

1. Look at the note symbol in the sheet music and compare it to the note symbols in Figure 9-2. The flag on the tail means it's an eighth note. The number next to the symbol, 8, is the duration value of an eighth note used by BIRTHDAY.

2. The head of the note's placement between the middle of the first and second lines on the lower F-clef staff means it is a G note in the third octave. Look up G3 in Figure 9-1 and find the pitch value, 196.

The DATA statement at 610 in BIRTHDAY has these same two values. The sound routine uses them to create the musical sound played when you sing the *Hap* of *Happy*.

To make it easier to convert sheet music, the MUSICMAKER program looks up the duration and pitch values for you. You just enter the letter shown in lower case in Figure 9-2 that represents the duration symbol and the pitch as a combination of the letter, sharp symbol (when required) and the octave number. For example, to enter the eighth note G of the third octave you enter the characters:

eg3

The next note is the same note, an eighth note with a pitch of G3.

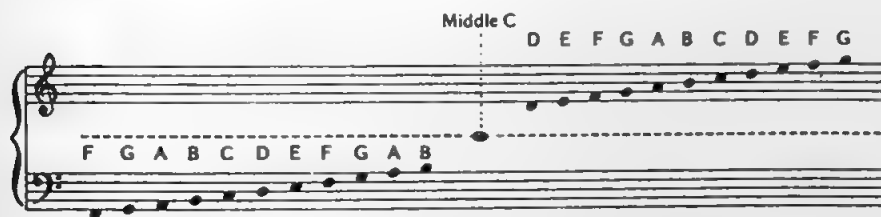


FIGURE 9-9 Musical Staves.

HAPPY BIRTHDAY TO YOU!

Mildred J. Hill
Patty S. Hill

Brightly *mf*

G C Am G D7 G

Hap-py Birth-day to you, Hap-py Birth-day to you, Hap-py

C Cm6 G D7 G

Birth-day, dear _____, Hap-py Birth-day to you!

The sheet music is written for piano and voice. It features a treble and bass clef for the piano accompaniment and a single treble clef for the voice. The key signature has one sharp (F#) and the time signature is 4/4. The music is divided into three systems. The first system is an instrumental introduction marked 'Brightly' and 'mf', with a piano accompaniment in the left hand and a melody in the right hand. The second system contains the first line of the song: 'Hap-py Birth-day to you, Hap-py Birth-day to you, Hap-py'. The third system contains the second line: 'Birth-day, dear _____, Hap-py Birth-day to you!'. Chord symbols (G, C, Am, D7, Cm6) are placed above the staff to indicate the harmonic structure. The lyrics are written below the staff.

• Here insert the name of the one celebrating.

© 1935 by Sammy-Berchard Music
division of Birch Tree Group Ltd
Princeton, New Jersey Copyright renewed
This arrangement © 1979
All rights reserved Printed in U.S.A.
ISBN 0-87467-404-1

FIGURE 9-10 "Happy Birthday" Sheet Music.

PITCH

A5
G5
F5
E5
D5
C5
B4
A4
G4
F4
E4
D4
C4
B3
A3
G3
F3
E3
D3
C3
B2
A2
G2
F2
E2



Hap-py birth- day to you, hap-py

PITCH

A5
G5
F5
E5
D5
C5
B4
A4
G4
F4
E4
D4
C4
B3
A3
G3
F3
E3
D3
C3
B2
A2
G2
F2
E2



birth-day to you! Hap-py birth-day, dear

PITCH

A5
G5
F5
E5
D5
C5
B4
A4
G4
F4
E4
D4
C4
B3
A3
G3
F3
E3
D3
C3
B2
A2
G2
F2
E2



name, hap-py birth-day to you!

The next three notes, without flags on the tail, are the quarter notes A3, G3 and C4. When you use MUSICMAKER you will enter these notes as:

qa3

qg3

qc4

The nonsolid head of the sixth note, B3, marks it as a half note. The second *happy birthday* has the same first four notes as the first, but the *to you* notes are played a half step higher as D4 and C4. The two G3 eighth notes accompany the third *happy*, but a different and higher-pitch pattern of quarter notes, G4, E4 and C4, emphasizes the *birthday dear*.

A quarter note, B3, begins the person's name. The *dot* after the A3, which ends the name, means increase the duration by half. We've defined the normal duration for a quarter note as 16: the *dot* increases the duration by half of 16 which is 8, making the total relative duration 24. A double dot would increase the duration by half of the first increase. For example, if there were two dots after the quarter note, the duration would increase to 28 as shown below:

16	original duration of quarter note
8	half of 16 indicated by first dot
4	half of 8 indicated by second dot
<u>28</u>	total duration of quarter note with two dots

For MUSICMAKER you write the dotted quarter B in the third octave as

q.b3

If the sheet music showed two dots, use the colon in place of the period. For example,

q:b3

Following the name, two F4 eighth notes, higher in pitch than the two G3 eighth notes, emphasize the *happy*. The next three quarter notes E4, C4 and D4, accompany *birthday to*. The last half note, C4, a middle C, has a dot so it plays for a 48 duration—32, the regular half-note duration, plus half of 32 or 16.

The DATA statements in lines 610 through 666 define the duration and pitch values for these notes and show the MUSICMAKER format for entering the notes.

CONVERTING OTHER SHEET MUSIC

The “Happy Birthday” sheet music in the *Companion* shows only the melody notes. Most sheet music has other notes written directly below some of the melody notes. Playing these notes at the same time as the melody notes enhances the sound. For simplicity’s sake, the MUSIC-MAKER plays only one note at a time, so you should only enter the note code for the top melody note. For example, in the music fragment in Figure 9-11 you enter only the two top eighth notes and not the quarter note below them.

Instead of writing the flat or sharp symbol after the note, composers often indicate notes on a particular staff line as a sharp or flat by writing a sharp or flat symbol at the beginning of the line. For example, in the music fragment in Figure 9-12, the sharp symbol (#) at the beginning of the line means play both F4 notes as F#4.

“Happy Birthday” has no sharp or flat notes. For MUSICMAKER, you enter flats as sharps; for example, B flat is A sharp. For a whole note of A sharp in the third octave, you would enter wa#3. The w indicates



FIGURE 9-11 Chord.



FIGURE 9-12 Sharp at Beginning of Bar.

a whole note duration, the a# the A sharp step in the octave, and the 3 the octave number.

While the *Companion* has given you the essential information you need to convert the melody of most sheet music into note codes that ADAM can read and play, music involves more notation than explained here. If you want to learn more about music and musical notation, the *Companion* recommends you read *What Makes Music Work*, a self-teaching guide by Seyer, Novick and Harmon, published by John Wiley & Sons.

MUSICMAKER FUNCTIONS

While BIRTHDAY can play music, to change it into a real tool for entering the musical codes described above you need to add functions similar to those in PICMAKER. You need the following functions:

ENTER: to enter a list of note codes, the music you want ADAM to play.

PLAY: to tell the program to play the music you have entered.

CURSOR UP AND DOWN: to select a particular note code that you want to change or delete, or to indicate where you want to insert a note code.

ADD: to insert a note code, making room in the list by shifting all the following codes down.

DELETE: to delete a note code, because it doesn't sound right or is incorrectly entered. After deletion you shift the list of note codes up, to fill in the empty space.

STORE/GET: so you can save the note codes as a file on tape or disk and get them back whenever you want to play or change them.

TEMPO: so you can change the speed the music plays. Some music sounds better played faster or slower. Sheet music indicates the tempo of the music by using the Italian words *allegro* for fast, *moderato* for medium and *adagio* or *largo* for slow.

PRINT: so you can list the music codes and values for duration and pitch. To play music in other programs, include the **BIRTHDAY** program as a subroutine, as well as the **DATA** statements that define the duration and pitch. The **PRINT** function gives you the values you need for the **DATA** statements.

Like **PICMAKER**, **MUSICMAKER** uses function keys to select the function. Figure 9-13 shows the screen layout used by **MUSICMAKER** to display the function prompts on the bottom two lines. Line 21 displays the action message and the note number pointed to by the cursor. Line 22 displays error messages and is used to enter data such as the filename used by the **STORE/GET** function. The top 19 lines display 19 of the note codes in the list of codes that defines music.

CHANGING BIRTHDAY INTO MUSICMAKER

Load **BIRTHDAY**, for you're going to delete and add statements to change it into **MUSICMAKER**. Change the program name in the statement 10 **REM** to **MUSICMAKER**. Delete the **DATA** statements that define "Happy Birthday" and the statements that load their duration

TEXT MODE SCREEN

		COLUMNS																																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					
LINES	1																																				
	2	1																																			
	3	2																																			
	4	3																																			
	5	4																																			
	6	5																																			
	7	6																																			
	8	7																																			
	9	8																																			
	10	9																																			
	11	10																																			
	12	11																																			
	13	12																																			
	14	13																																			
	15	14																																			
	16	15																																			
	17	16																																			
	18	17																																			
	19	18																																			
	20	19																																			
	21	press a key																																			
	22	did =																																			
	23	I																																			
	24	ENTER																																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					

FIGURE 9-13 Musicmaker Screen Layout.

and pitch values into the *music%* array. MUSICMAKER doesn't need them because it will store the music values as a file on the tape or disk. You type

DEL 610,820

Add the following statements that create the main loop and put up the subroutine studs for each function key, just as you did with PICMAKER. In fact, the statements are almost identical; only the function names have changed.

```

10 REM    musicmaker
100 DIM music$(max), music%(max, 2), note$(12), frequency%(96)
110 DIM func%(11)
900 GOSUB 8000
1000 REM          music main loop
1010 VTAB 21: HTAB 1: PRINT "press a key": TAB(18): "note=": music%
1020 HTAB column%: VTAB row%
1100 GET key$: key% = ASC(key$)
1220 IF key% = 3 THEN STOP
1400 FOR func = 1 TO 11
1410 IF func%(func) = key% THEN 1600
1420 NEXT func
1440 VTAB 22: HTAB 1
1450 PRINT beep$: "WRONG KEY! PRESS FUNCTION KEY"
1550 GOTO 1000
1600 HTAB 1: VTAB 21: PRINT SPC(30): PRINT SPC(30):
1610 HTAB 1: VTAB 22: PRINT "did=":
1620 ON func GOSUB 2000, 2400, 2800, 3000, 3200, 3400, 3600, 4000, 4400, 5000,
5600
1630 IF exit% THEN END
1990 GOTO 1000
2000 REM          move cursor up
2100 PRINT "up"
2190 RETURN
2400 REM          move cursor down
2500 PRINT "down"
2690 RETURN
2800 REM          enter note
2810 PRINT "enter note": beep$:
2990 RETURN
3000 REM          play
3200 REM tempo
3210 PRINT "set tempo"
3390 RETURN
3400 REM add note
3585 PRINT "add note"
3590 RETURN
3600 REM delete note
3785 PRINT "delete note"
3790 RETURN
4000 REM          order/command
4100 PRINT "command":
4110 HTAB 1: VTAB 21: INPUT "command=": cmd$
4120 IF cmd$ = "end" OR cmd$ = "END" THEN exit% = true%
4390 RETURN
4400 REM          print
4410 PRINT "print"
4990 RETURN
5000 REM          store/get
5100 PRINT "store/get"
5590 RETURN

```



```

5600 REM                      new clear screen
5610 TEXT: HONE
5620 PRINT TAB(5); "DDNNO";
5630 HTAB 1: VTAB 23
5640 PRINT " I "; " II "; " III ";
5670 PRINT " IV "; " V "; " VI ";
5710 PRINT "ENTER"; " PLAY"; "tempo";
5740 PRINT " ADD "; " DEL "; "ORDER";
5750 FOR ms = 1 TO max: music$(ms) = " ": NEXT ms
5760 music% = 1
5770 ms% = music%
5800 REM  display notes
5810 last% = ms%+18
5820 IF last% > max% THEN last% = max%
5830 HTAB 1: VTAB 2
5840 FOR ms = ms% TO last%
5850 PRINT ms; TAB(5); music$(ms)
5860 NEXT ms
5990 RETURN
6000 REM                      initialization
6010 register% = 0
6030 true% = 1: false% = 0
6040 beep$ = CHR$(7)+CHR$(7): d$ = CHR$(4)
6600 DATA 160,162
6610 REM up,down
6620 DATA 129,130,131,132,133,134
6630 REM enter,play,tempo,add,del,order
6640 DATA 149,147,150
6650 REM print,store/get,clear
6670 FOR func = 1 TO 11
6680 READ func%(func)
6690 NEXT func
6700 GOSUB 5600: REM                      clear screen
6710 GOSUB 16700
6999 RETURN

```

FIGURE 9-14 MUSICMAKER Program.

SAVE the program and then RUN it to test that each function key GOSUBs to the proper subroutine.

Scrolling Through The List

Most songs you want to enter into ADAM using MUSICMAKER have more than 19 notes, the maximum you can display on the screen at one time. For example, "Happy Birthday," which is a short song, requires 25 notes. SmartWriter has the same problem when the letter or report has more lines than the screen can hold. To solve this problem, it lets you use the UP and DOWN cursor keys to scroll the screen up and down, displaying the lines you want to view. MUSICMAKER also allows you to use the UP and DOWN cursor keys to display the part of the note list you want to view; however, it scrolls 10 lines at a time instead of one. Like SmartWriter it allows you to change, delete and insert anything displayed.

Enter the statements in Figure 9-15.


```

2120 IF music% = 1 THEN PRINT beep$: RETURN
2130 music% = music%-1
2140 IF row% > 2 THEN row% = row%-1: RETURN
2150 ms% = music%-8: row% = row%+8: GOSUB 5800: RETURN
2510 IF music% = max% THEN PRINT beep$: RETURN
2520 music% = music%+1
2530 IF row% < 20 THEN row% = row%+1: RETURN
2540 ms% = music%-10: row% = row%-8: GOSUB 5800: RETURN

```

FIGURE 9-15 Cursor Scroll.

MUSICMAKER stores the note codes as the string array *music\$*. The scroll routine displays 19 note codes from this array beginning at the subscript defined by the variable *ms%*.

Initialization

To convert the duration and note letters and octave numbers of the note code into values for the sound chip, you need conversion tables. MUSICMAKER reads the letter and corresponding number values from DATA statements and stores them into the arrays *note\$* and *frequency%*, and *duration\$* and *duration%*. DATA statements define only the frequency values for the second octave. The FOR . . . NEXT loop uses the fact that each of the values for the next octave is double the current octave's values to calculate the values for octaves 3 through 6. Enter the statements in Figure 9-16.

```

7000 REM      note values
7020 DATA "c","c#", "d","d#", "e","f","f#", "g","g#", "a","a#", "b"
7110 DATA 65.405,69.295,73.415,77.783,82.408,87.308
7120 DATA 92.498,97.998,103.83,110,116.64,123.47
7440 DATA "w",64,"h",32
7450 DATA "q",16,"e",8
7460 DATA "s",4,"t",2
8100 FOR note = 1 TO 12
8120 READ note$(note)
8140 NEXT note
8200 FOR note = 1 TO 12
8204 READ frequency
8210 FOR octave = 0 TO 5: REM      octave 2 to 7
8212 fs = note+((octave)*12)
8214 frequency%(fs) = frequency*(2^octave)
8220 NEXT octave
8240 NEXT note
8250 FOR duration = 1 TO 6
8260 READ duration$(duration), duration%(duration)
8270 NEXT duration
8280 duration = 1000

```

FIGURE 9-16 Initialization.

Editing the Note Codes

The format of the note code is D.N#O. While the codes are shown in capitals, MUSICMAKER requires you to enter them as lowercase letters.

D Stands for the duration letter code, which must be one of the letters (W,H,Q,E,S,T) from Figure 9-2.

- Use the period or colon, when required, to indicate the increase in duration of the pitch.

N Stands for the note and must be one of the letters A through G.

Use the sharp symbol, when required, to indicate a pitch one half step higher than the letter note.

O Stands for the octave. The octave range is from 2 through 6. The letter note, sharp symbol and the octave number define the pitch of the musical sound.

The following statements allow you to enter the note. The routine edits the code. If invalid, it displays one of these messages:

INVALID DURATION

INVALID NOTE

INVALID OCTAVE

If valid, it plays the note. It stores the note code in the *music\$* array and the duration and pitch values in the *music%* array.

After entering the statements in Figure 9-17, you can enter note codes and play them back using the play function.

```

2805 HTAB 1: VTAB 21
2815 VTAB row%: HTAB column%: PRINT SPC(10)
2820 VTAB row%: HTAB column%: INPUT " "; field$
2840 IF LEN(field$) = 0 THEN RETURN
2850 GOSUB 3800: REM      edit input
2860 IF error% THEN 2800
2870 GOSUB 2900: REM      compute and store values
2880 GOSUB 6000: REM      play note
2890 RETURN
2900 subscript% = note% + ((octave% - 2) * 12)
2905 pitch% = frequency%(subscript%)
2910 music%(music%, frequency%) = pitch%
```



```

2920 duration = duration+duration*dot+duration*ddot
2930 music%(music%, time%) = duration
2940 music$(music%) = field$
2950 GOSUB 2510: REM          down
3040 FOR music = music% TO max
3050 pitch% = music%(music, frequency%)
3060 duration = music%(music, time%)
3070 IF pitch% AND duration THEN GOSUB 6000
3080 NEXT music
3800 REM          edit input
3801 error% = true%: REM          assume error
3803 work$ = field$
3805 size% = LEN(work$)
3807 n% = 1: d% = 1: dot = 0: ddot = 0
3810 IF size% = 5 THEN d% = 2: n% = 2
3815 o$ = RIGHT$(work$, 1)
3820 HTAB 1: VTAB 22: REM          for error message
3830 IF VAL(o$) THEN size% = size%-1: work$ = LEFT$(work$, size%)
3840 IF RIGHT$(work$, 1) = "#" THEN n% = 2
3850 dot$ = MID$(work$, 2, 1)
3860 IF dot$ = "." THEN d% = 2: dot = .5
3862 IF dot$ = ":" THEN d% = 2: ddot = .25
3865 IF n%+d% >= size% THEN error% = 3960
3870 note$ = RIGHT$(work$, n%): duration$ = LEFT$(work$, 1)
3880 FOR duration = 1 TO 6
3890 IF duration$ = duration$(duration) THEN 3920
3899 NEXT duration
3910 PRINT "INVALID DURATION": RETURN
3920 duration = duration%(duration)
3930 FOR note = 1 TO 12
3935 IF note$ = note$(note) THEN note% = note: GOTO 3950
3939 NEXT note
3940 PRINT "INVALID NOTE": RETURN
3950 IF o$ >= "2" AND o$ <= "7" THEN octave% = VAL(o$): GOTO 3970
3960 PRINT "INVALID OCTAVE": RETURN
3970 error% = false%: REM          passed all checks
3990 RETURN

```

FIGURE 9-17 Edit and Convert.

SAVE the program and then RUN it, entering the first six notes in "Happy Birthday":

1. eg3
2. eg3
3. qa3
4. qg3
5. qc4
6. hb3

Happy Birthday to you!

Don't enter any more notes, because you still need the *store/get* subroutine to store the note codes in a file.

Saving the Music

LOAD the STOREGET program you wrote in Chapter 7. The next statements modify it for use by MUSICMAKER. After LOADING the program, delete the following statements.

```

10 REM musicstget
100 DIM music$(100)
120 ds = CHR$(4)
200 DATA ec,ed,ea
300 FOR ms = 1 TO 3
310 READ music$(ms)
320 NEXT ms
1000 GOSUB 5300: REM save song
1050 GR: VTAB 23: HTAB 1: PRINT "loading file":
1100 GOSUB 5200: REM load song
1200 END
5000 REM store/get picture
5100 PRINT "store/get"
5110 HTAB 1: HTAB 21: INPUT "STORE/GET(s/g)?": ans$
5115 HTAB 1: VTAB 22: PRINT SPC(30):
5120 IF ans$ = "S" OR ans$ = "s" THEN 5300
5130 IF ans$ = "G" OR ans$ = "g" THEN 5200
5140 HTAB 1: VTAB 22: PRINT "Not s or g": RETURN
5200 REM load song
5210 op$ = "get"
5220 GOSUB 5550: IF none% THEN RETURN
5230 ONERR GOTO 5285
5240 PRINT d$: "open "; name$
5250 PRINT d$: "read "; name$
5260 FOR ms = 1 TO 100
5265 INPUT music$(ms)
5267 CLRERR: REM found file
5280 NEXT ms
5282 PRINT d$: "close "; name$
5283 GOSUB 5760: REM display notes
5284 RETURN
5285 REM onerr routine
5286 CLRERR: GOSUB 5580
5290 PRINT d$: "close "; name$
5293 PRINT d$: "delete "; name$
5299 RETURN
5300 REM save song
5310 op$ = "store"
5320 GOSUB 5550: IF none% THEN RETURN
5340 PRINT d$: "open "; name$
5350 PRINT d$: "write "; name$
5360 FOR ms = 1 TO 100
5365 PRINT music$(ms)
5370 NEXT ms
5385 PRINT d$: "close "; name$
5387 HTAB 1: VTAB 22: PRINT "SONG "; name$: " SAVED"
5390 RETURN
5550 REM input song name
5555 none% = false%
5557 HTAB 1: VTAB 21: PRINT SPC(30):
5558 HTAB 1: VTAB 21: PRINT op$
5560 INPUT "song name=": name$
5570 IF LEN(name$) > 0 THEN RETURN
5575 TEXT
5580 HTAB 1: VTAB 22: PRINT "NO SONG "; name$

```

FIGURE 9-18 MUSICSTGET.

Test the program, making sure that you can write elements from the *music\$* array to a file and back again. To MONitor the commands

and data,

You type MON c,i,o

When you're satisfied the program works correctly insert statements,

5283 GOSUB 5760: REM first note

5273 GOSUB 3800: REM edit

5275 GOSUB 2900: REM load

SAVE it as MUSICSTGET. Delete the test statements and SAVE again as MUSICTEMP. The use SmartWRITER to load MUSICMAKER and merge in MUSICTEMP. Then STORE MUSICMAKER. After reloading BASIC, use the LOAD command to bring MUSICMAKER into memory.

Inserting and Deleting

When you've missed a note or forgotten to enter it, you can insert it by pressing the IV function key. The program inserts a note in the *music\$* array by shifting all elements from the current note, defined by *music%*, to the end of the table down one position. It blanks out the current display line and lets you enter a new note code. For example, if you had two notes in a three-element *music\$* array and you wanted to insert another eg3 note code before the qa3, you would use the cursor keys to position the cursor to the second note, which corresponds to the second element in the *music\$* array. This sets *music%* to the value 2. When you press function key IV for ADD, the program shifts the elements down as shown below.

	Array element	Before shift	After shift
	1	eg3	eg3
current note →	2	qa3	
	3		qa3

The current line has a blank, and you're directed to enter a note.

To delete a note, you press the V function. The program deletes the element by shifting all elements up and blanking on the last element in array. For example, suppose you changed your mind and really only wanted the two original notes eg3 and qa3. The program deletes the second note as shown below.

	Array element	Before shift	After shift
	1	eg3	eg3
current note →	2	eg3	qa3
	3	qa3	

The note code at the cursor location was deleted, the last note code was shifted up, and the last element in the array was blanked out so the last note code wasn't repeated twice.

Enter the following statements and test that the program works as described.

```

3410 FOR ms = max-1 TO music% STEP -1
3420 music$(ms+1) = music$(ms)
3440 music%(ms+1, time%) = music%(ms, time%)
3450 music%(ms+1, frequency%) = music%(ms, frequency%)
3460 NEXT ms
3470 music$(music%) = ""
3475 music%(music%, time%) = 0: music%(music%, frequency%) = 0
3480 ms% = music%-row%+2: GOSUB 5800
3610 FOR ms = music% TO max-1
3620 music$(ms) = music$(ms+1)
3640 music%(ms, frequency%) = music%(ms+1, frequency%)
3650 music%(ms, time%) = music%(ms+1, time%)
3660 NEXT ms
3670 ms% = music%-row%+2: GOSUB 5800

```

FIGURE 9-19 Insert and Delete.

Changing the Tempo

When you press function key III, line 22 displays the prompt
ENTER TEMPO=

you can now change the value in the variable *tempo*, which the program initializes to 20. A low number like 5 speeds up the music, while a high number like 40 slows it down.

Enter the following two statements and you can experiment yourself.

```

3220 HTAB 1: VTAB 22: INPUT "temp="; tempo$
3230 IF LEN(tempo$) <> 0 THEN tempo = VAL(tempo$)

```

FIGURE 9-20 Tempo.

PRINTING MUSIC CODES

After entering the programs in this book, you know how important it is to have a printed listing of a BASIC program. It much easier to work with a piece of paper that you can scribble on and carry with you. With large programs and music compositions, the screen displays only a portion of the complete instructions. With a printed listing you can see the whole thing, which is often helpful when you're looking for an error or deciding where to make a change in the music or program. When you press the PRINT key, the following statements print not only the music codes in the *music\$* array, but also the values for the note pitch in Hz and relative duration. You can put these values in the DATA statements of other programs you want to play music that have the BIRTHDAY program as the sound subroutine.

```

4420 PR #1: FOR ms = music% TO max
4430 IF lctr = 0 THEN GOSUB 4600
4435 IF LEN(music$(ms)) = 0 THEN 4450
4440 PRINT ms: TAB(6): music$(ms): TAB(13):
4445 PRINT music%(ms, time%): TAB(20): music%(ms, frequency%)
4450 NEXT ms: PR #0
4455 GOSUB 5600: REM      clear screen
4460 ms% = music%-row%+2: GOSUB 2600: REM      display notes
4490 RETURN
4600 REM      heading
4610 lctr = 60
4620 IF NOT first THEN FOR l = 1 TO 6: PRINT: NEXT l: first = true%
4630 PRINT "NOTE CODE DURATION PITCH"

```

FIGURE 9-21 Print.

FINAL LISTING

On the following pages you have a complete listing of the MUSICMAKER program (Figure 9-22). If your program doesn't work correctly, match it line by line with this listing. You have probably left a few statements out, or misspelled a variable.


```

5 LOMEM :29000
10 REM      musicmaker 1/28/84
90 max = 100: maxt = max
100 DIM music$(max), musict(max, 2), note$(12), frequencyt(96,
110 DIM funct(11)
210 timet = 1: frequencyt = 2
500 GOSUB 8000
1000 REM      music main loop
1010 VTAB 21: HTAB 1: PRINT "press a key"; TAB(10); "note="; musict
1020 HTAB columnt: VTAB rowt
1100 GET key$: keyt = ASC(key$)
1220 IF keyt = 3 THEN STOP
1400 FOR func = 1 TO 11
1410 IF funct(func) = keyt THEN 1600
1420 NEXT func
1440 VTAB 22: HTAB 1
1450 PRINT beep$: "WRONG KEY! PRESS FUNCTION KEY"
1550 GOTO 1000
1600 HTAB 1: VTAB 21: PRINT SPC(30): PRINT SPC(30):
1610 HTAB 1: VTAB 22: PRINT "did=":
1620 ON func GOSUB 2000, 2400, 2800, 3000, 3200, 3400, 3600, 4000, 4400, 5000,
5600
1630 IF exitt THEN END
1990 GOTO 1000
2000 REM      move cursor up
2100 PRINT "up"
2120 IF musict = 1 THEN PRINT beep$: RETURN
2130 musict = musict-1
2140 IF rowt > 2 THEN rowt = rowt-1: RETURN
2150 mat = musict-8: rowt = rowt+8: GOSUB 5800: RETURN
2190 RETURN
2400 REM      move cursor down
2500 PRINT "down"
2510 IF musict = maxt THEN PRINT beep$: RETURN
2520 musict = musict+1
2530 IF rowt < 20 THEN rowt = rowt+1: RETURN
2540 mat = musict-10: rowt = rowt-8: GOSUB 5800: RETURN
2590 RETURN
2800 REM      enter note
2805 HTAB 1: VTAB 21
2810 PRINT "enter note": beep$:
2815 VTAB rowt: HTAB columnt: PRINT SPC(10)
2820 VTAB rowt: HTAB columnt: INPUT " "; field$
2840 IF LEN(field$) = 0 THEN RETURN
2850 GOSUB 3800: REM      edit input
2860 IF errort THEN 2800
2870 GOSUB 2900: REM      compute and store values
2880 GOSUB 6000: REM      play note
2900 RETURN
2900 subscriptt = note$(octavet-2)*12)
2905 pitcht = frequencyt(subscriptt)
2910 musict(music, frequencyt) = pitcht
2920 duration = duration+duration*dot+duration*ddot
2930 musict(music, timet) = duration
2940 music$(music) = field$
2950 GOSUB 2510: REM      down
2990 RETURN
3000 REM      play
3040 FOR music = musict TO max
3050 pitcht = musict(music, frequencyt)
3060 duration = musict(music, timet)
3070 IF pitcht AND duration THEN GOSUB 6000
3080 NEXT music
3185 PRINT "play"
3190 RETURN
3200 REM      set tempo
3210 PRINT "set tempo"
3220 HTAB 1: VTAB 22: INPUT "tempo="; tempo$
3230 IF LEN(tempo$) <> 0 THEN tempo = VAL(tempo$)
3390 RETURN
3400 REM      add note
3410 FOR ms = max-1 TO musict STEP -1
3420 music$(ms+1) = music$(ms)
3440 musict(ms+1, timet) = musict(ms, timet)
3450 musict(ms+1, frequencyt) = musict(ms, frequencyt)
3460 NEXT ms
3470 music$(music) = ""
3475 musict(music, timet) = 0: musict(music, frequencyt) = 0
3480 mat = musict-rowt+2: GOSUB 5800

```



```

3585 PRINT "add note"
3590 RETURN
3600 REM          delete note
3610 FOR ms = music% TO max-1
3620 music$(ms) = music$(ms+1)
3640 music%(ms, frequency%) = music%(ms+1, frequency%)
3650 music%(ms, time%) = music%(ms+1, time%)
3660 NEXT ms
3670 ms% = music%-row%+2: GOSUB 5800
3785 PRINT "delete note"
3790 RETURN
3800 REM          edit input
3801 error% = true%: REM          assume error
3803 work$ = field$
3805 size% = LEN(work$)
3807 n% = 1: d% = 1: dot = 0: ddot = 0
3810 IF size% = 5 THEN d% = 2: n% = 2
3815 o$ = RIGHT$(work$, 1)
3820 HTAB 1: VTAB 22: REM          for error message
3830 IF VAL(o$) THEN size% = size%-1: work$ = LEFT$(work$, size%)
3840 IF RIGHT$(work$, 1) = "#" THEN n% = 2
3850 dot$ = MID$(work$, 2, 1)
3860 IF dot$ = "." THEN d% = 2: dot = .5
3862 IF dot$ = ":" THEN d% = 2: ddot = .25
3865 IF n%+d% >= size% THEN error% = 3960
3870 note$ = RIGHT$(work$, n%): duration$ = LEFT$(work$, 1)
3880 FOR duration = 1 TO 6
3890 IF duration$ = duration THEN 3920
3899 NEXT duration
3910 PRINT "INVALID DURATION": RETURN
3920 duration = duration%(duration)
3930 FOR note = 1 TO 12
3935 IF note$ = note$(note) THEN note% = note: GOTO 3950
3939 NEXT note
3940 PRINT "INVALID NOTE": RETURN
3950 IF o$ >= "2" AND o$ <= "7" THEN octave% = VAL(o$): GOTO 3970
3960 PRINT "INVALID OCTAVE": RETURN
3970 error% = false%: REM          passed all checks
3990 RETURN
4000 REM          order/command
4100 PRINT "command":
4110 HTAB 1: VTAB 21: INPUT "command=": cmd$
4120 IF cmd$ = "end" OR cmd$ = "END" THEN exit% = true%
4390 RETURN
4400 REM          print
4410 PRINT "print"
4420 PR #1: FOR ms = music% TO max
4430 IF lctr = 0 THEN GOSUB 4600
4435 IF LEN(music$(ms)) = 0 THEN 4450
4440 PRINT ms: TAB(6): music$(ms): TAB(13):
4445 PRINT music%(ms, time%): TAB(20): music%(ms, frequency%)
4450 NEXT ms: PR #0
4455 GOSUB 5600: REM          clear screen
4460 ms% = music%-row%+2: GOSUB 2600: REM          display notes
4490 RETURN
4600 REM          heading
4610 lctr = 60
4620 IF NOT first THEN FOR l = 1 TO 6: PRINT: NEXT l: first = true%
4630 PRINT "NOTE CODE DURATION PITCH"
4690 RETURN
4990 RETURN
5000 REM          store/get
5100 PRINT "store/get"
5110 HTAB 1: VTAB 21: INPUT "STORE/GET(s/g)?": ans$
5115 HTAB 1: VTAB 22: PRINT SPC(30):
5120 IF ans$ = "S" OR ans$ = "s" THEN 5300
5130 IF ans$ = "G" OR ans$ = "g" THEN 5200
5140 HTAB 1: VTAB 22: PRINT "Not s or g": RETURN
5200 REM          load song
5210 op$ = "get"
5220 GOSUB 5550: IF none% THEN RETURN
5230 ONERR GOTO 5285
5240 PRINT d$: "open ": name$
5250 PRINT d$: "read ": name$
5260 FOR ms = 1 TO 100
5265 INPUT " ": field$
5267 CLRERR: REM          found file
5270 duration = 0
5271 IF LEN(field$) = 0 THEN GOSUB 2930: GOTO 5280

```



```

5273 GOSUB 3800: REM edit
5275 GOSUB 2900: REM load
5280 NEXT ms
5282 PRINT d$: "close "; name$
5283 GOSUB 5760: REM first note
5284 RETURN
5285 REM onerr routine
5286 CLRERR: GOSUB 5580
5290 PRINT d$: "close "; name$
5293 PRINT d$: "delete "; name$
5299 RETURN
5300 REM save picture
5310 op$ = "store"
5320 GOSUB 5550: IF none THEN RETURN
5340 PRINT d$: "open "; name$
5350 PRINT d$: "write "; name$
5360 FOR ms = 1 TO 100
5365 PRINT music$(ms)
5370 NEXT ms
5385 PRINT d$: "close "; name$
5387 HTAB 1: VTAB 22: PRINT "SONG "; name$; " SAVED"
5390 RETURN
5550 REM input song name
5555 none = false
5557 HTAB 1: VTAB 21: PRINT SPC(30);
5558 HTAB 1: VTAB 21: PRINT op$;
5560 INPUT "song name="; name$
5570 IF LEN(name$) > 0 THEN RETURN
5580 HTAB 1: VTAB 22: PRINT "NO SONG "; name$
5585 none = true
5590 RETURN
5600 REM new clear screen
5610 TEXT: HOME
5620 PRINT TAB(5); "DDNNO"
5630 HTAB 1: VTAB 23
5640 PRINT " I "; " II "; " III ";
5670 PRINT " IV "; " V "; " VI ";
5710 PRINT "ENTER"; " PLAY"; "tempo";
5740 PRINT " ADD "; " DEL "; "ORDER";
5750 FOR ms = 1 TO max: music$(ms) = "": NEXT ms
5760 music% = 1
5770 ms% = music%
5780 row% = 2: column% = 5
5800 REM display notes
5810 last% = ms%+18
5820 IF last% > max THEN last% = max%
5830 HTAB 1: VTAB 2
5840 FOR ms = ms% TO last%
5850 PRINT ms; TAB(5); music$(ms)
5860 NEXT ms
5990 RETURN
6000 REM play note
6005 pitch% = mhzrtz/(32*pitch%)
6010 second% = pitch%/16
6020 first% = 128+register%+(pitch%-second%*16)
6030 POKE chip%, first%: CALL sound%
6040 POKE chip%, second%: CALL sound%
6050 POKE chip%, loud%: CALL sound%
6100 FOR delay = 1 TO duration*tempo: NEXT delay
6120 POKE chip%, quiet%: CALL sound%
6300 RETURN
7000 REM note value
7020 DATA "c","c#", "d","d#", "e","e#", "f","f#", "g","g#", "a","a#", "b"
710 DATA 65.405,69.295,73.415,77.783,82.408,87.308
7120 DATA 92.498,97.998,103.83,110,116.64,123.47
7440 DATA "w",64,"h",32
7450 DATA "q",16,"e",8
7460 DATA "s",4,"t",2
8000 REM initialization
8010 register% = 0
8030 true% = 1: false% = 0
8040 beep$ = CHR$(7)+CHR$(7): d$ = CHR$(4)
8100 FOR note = 1 TO 12
8120 READ note$(note)
8140 NEXT note
8200 FOR note = 1 TO 12
8204 READ frequency
8210 FOR octave = 0 TO 5: REM octave 2 to 7
8212 fs = note+((octave)*12)

```



```

8214 frequency%(fs) = frequency*(2^octave)
8220 NEXT octave
8240 NEXT note
8250 FOR duration = 1 TO 6
8260 READ duration$(duration), duration%(duration)
8270 NEXT duration
8280 duration = 1000
8600 DATA 160,162
8610 REM up,down
8620 DATA 129,130,131,132,133,134
8630 REM enter,play,tempo,add,del,order
8640 DATA 149,147,150
8650 REM print,store/get,clear
8670 FOR func = 1 TO 11
8680 READ func%(func)
8690 NEXT func
8700 GOSUB 5600: REM clear screen
8710 GOSUB 16700
9999 RETURN
16700 REM sound
16705 DATA 58,102,109,211,255,201
16720 loud% = 144: quiet% = 159
16730 sound% = 28000
16740 chip% = 28006
16750 mhertz = 3597000
16760 tempo = 20
16810 FOR address = sound% TO sound%+5
16820 READ byte%
16830 POKE address, byte%
16840 NEXT address
16990 RETURN

```

FIGURE 9-22 MUSICMAKER Final Listing.

ENHANCEMENTS

While the MUSICMAKER has many useful features, the following were left out. If you have the time and energy, you may want to add them yourself. Be aware, however, that these are not simple changes and will require some experimentation, thought and program design work.

Change the program to accept the flat symbol in addition to the sharp symbol. Use lowercase b to represent the flat symbol. Change the program to play two or three notes at the same time.

A Musical and Video Birthday Card

Now that you have the tools to draw pictures and play music, you might want to create a unique birthday card for your family and friends that only ADAM can deliver.

How to Write a Mailing-Label System in BASIC

You have seen how to use SmartWriter to store and retrieve data such as names and addresses, recipes and other lists of items. The SmartWriter technique, although simple, has some disadvantages.

- You have no prompting word to identify the data in the list and what order you must enter it. For example, SmartWriter doesn't prompt you with "ENTER FIRST NAME=". You must remember that you enter the person's first name and then the last name and not the other way around.
- You can't check the data for correctness or completeness. For example, the zip code should contain only the character 0-9, no letters or special symbols. If entered, it must have 5 or 9 digits. An error message should tell you that ABC is wrong, but it doesn't.
- If you have a lot of names, SmartWriter does not have enough room to store the entire document in memory. It can store only 65,536 characters, while a tape file can store 250,000 characters.
- You can only print the data in the order and form the data appears in the file. For example, if you store names, addresses and telephone numbers, you may want to print the data in two different formats: once as a 6-line mailing label and then as single-line name and telephone number directory. You can't do this with SmartWriter.

Coleco and other vendors provide generalized as well as specific data storage and retrieval software programs. In many instances software such as SmartFiler, when available, will meet your needs. But if you don't want to wait for or pay for this software, the

Companion explains how to write, in BASIC, your own name-and-address data storage and retrieval system. When you're finished writing and testing it, you'll be able to enter the names and addresses of your friends, relatives and business associates and then print mailing labels or a telephone directory.

FILE ORGANIZATION

ADAM provides two methods of organizing data—sequential and random. These were explained in Chapter 6. When you type a SmartWriter document you create a sequential file. Each line, which you end by pressing the RETURN key, is a record. The end-of-line or record symbol appears on the screen as a small triangle. One line follows another, line by line, creating a sequential file—the text document. In the PICMAKER and LETTRCHASE programs you learned to write a BASIC program using sequential-file-accessing commands to read and write a file. In the mailing system you will learn how to read and write records randomly.

A random file organization means that you can read or write records in any order. For example, the SmartWriter directory provides random retrieval of any document or file. Instead of having you read through every name and address, the *Companion's Little Black Book* mailing system allows you to read directly the information about anybody in the file using an identification number.

Name And Address List

In many cases, it's not what you know but who you know that benefits you. Keeping track of the names, addresses and telephone numbers of friends, relatives and business contacts is a must for everybody. If the information were on an ADAM data-pack cassette or diskette, think how easy it would be for you to do a mailing for:

Political action
Fund raising
Christmas cards

HOW YOU USE YOUR LITTLE BLACK BOOK

Find your address book. Examine it. You will find it consists of pages tabbed in alphabetical order. Rather than search through the entire book for a name, you use the tab to flip to the page of the person's "partial key"—the first letter of the last name. Then you only have to search a few names to find a match of the "full key"—the person's last and first name. For simplicity, the *Little Black Book* uses a number to identify the person. You can improve the system to keep a separate directory file of names in alphabetical order, so you can reference a person by name as well as number.

How else do you use your address book?

You add or write the name, address, zip code, telephone number and other information about new friends, relatives and acquaintances into the book.

When a person changes name, moves or gets a new telephone number you change or update it in your book.

Sometimes a person dies or you lose touch with a person. This time you want to delete or completely erase the information. Then you can use the space in the book for someone else's name.

When you want to announce a birth, invite people to a shower or wedding or send address-change notices, you want to scan all the names, selecting those you want to telephone or send a letter to. You identify the person selected based on information stored in your head about the person. To use ADAM effectively, you may want to store additional information about the person along with the name and address. By defining selection criteria, ADAM will search and select the desired names.

The study of the mailing-list process reveals at least two major processes:

1. File maintenance—random access to add, update, delete and display name and address information.
2. Sequential retrieval and selection to print mailing labels and a directory of names, addresses and identification numbers.

In addition, you will find it useful to keep control information about the file. You will want to know the name of the mailing list, the number of names in the file, the maximum you can store, today's date

and the date of last update. You may think of other information you want to store about the file. The *Little Black Book* provides a program to perform the process of displaying and updating this control information.

Since you want random-access capability, the *Little Black Book* has a program that formats the file to the record size and reserves space on the data-pack tape cassette or disk to store the potential maximum names.

While you could write a mailing system as one big program with main subroutines for each major process, the *Little Black Book* uses the more manageable approach of having separate programs for each major process. To tie the whole system together operationally, it provides a menu program that displays a list of all the major processes the system performs. When using the system, the operator just types RUN MAILMENU, which displays the list and selects the major process—file maintenance, print labels, print directory or display control information. Then the menu program will RUN the desired processing program. Figure 10-3 illustrates the *Little Black Book's* menu screen.

By now you have some programming skill. Rather than guide you through the development of each program as it did with PICMAKER, LETTRCHASE, and MUSICMAKER, the *Companion* describes each program, provides the screen layout design, a diagram of the program, and a listing in BASIC. Most books and magazines that publish program listings provide the same information. Plan to spend about 35 hours to enter all the *Little Black Book* programs, correct typing mistakes and test the system. When you complete each program, make sure you SAVE a copy of the program on a backup tape, just in case there's an accident with your original.

The *Companion's* mailing-label system consists of the following programs:

- MAILMENU—Choose the program you want to run
- MAILMAINT—Add, change and delete records
- MAILLABELS—Print labels
- MAILCONTRL—Display and update control information
- MAILFORMAT—Format file

The menu program provides for the MAILLIST program, which prints a directory listing, but you will have to design and write it yourself. The MAILLABELS program provides an example of how to do it.

Information To Store

Developing a data storage and retrieval system takes a lot of time and effort. That means you want to think about what information you want to store before you write the programs. Like changing the architectural plans to add another room after the house is built, it can be done but it costs more than if you had planned to do it during the design stage. So think of the different people you know and what information you want to store about them in the mailing file.

You receive a thank-you letter for a wedding gift from newlyweds who have moved to a new home. From the envelope you discover the following information:

- Husband's first name
- Wife's first name
- Couple's last name(s)
- Street address
- City
- State
- Zip code
- Telephone number

The plumber, who fixed your bathroom sink, gives you his business card on the way out of your house. It contains the following information:

- Company name
- Type of business
- Plumber's first and last name
- Title
- Street address
- City
- State
- Zip code
- Telephone number

Notice that while some of the data for the plumber is the same as the newlywed's, other data items differ. The newlyweds have two first names, while the plumber has a company name, type of business, and title. The record design must handle these differences. One technique has different record layouts for each type of record. You could have

one layout for the information about friends, another for relatives and a third for businesses. You may find this a desirable design, but it takes more programming than the *Companion* has space to demonstrate. The simpler approach uses *fields* in the record that store various data. For example, one field could store the plumber's company name or provide spaces for the newlyweds' first names.

The *Companion's* mailing-label system will print five lines of name and address information as shown below:

First and last name
 Second line
 Third line
 Fourth line
 City, state zip code

The second through fourth lines are each separate fields in the record. The street address should occupy the fourth line. You can use the other two lines to store the company name and title or any other address information. Blank lines between the first and the fifth will not be printed. However, if necessary, additional blank lines will be printed after the city and state to fill out the five-line labels. A sixth line prints a space to provide a line between labels.

In addition, the *Companion* provides room to store a status code, date added, identification codes, comments, and expansion. Figure 10-1 shows data-record layout and Figure 10-2 the control-record layout.

<i>Description</i>	<i>Size</i>	<i>From</i>	<i>To</i>	<i>Type</i>
Status character	1	1	1	alpha
First name	12	2	13	alpha
Last name	17	14	30	alpha
Address line 2	30	31	60	alpha
Address line 3	30	61	90	alpha
Address line 4	30	91	120	alpha
City	14	121	134	alpha
State	2	135	136	alpha
Zip code	5	137	141	numeric
Telephone number	10	142	151	numeric
Date added	8	152	159	alpha
Mail codes	8	160	167	alpha
Comment	20	168	187	alpha
Expansion space	12	188	199	alpha
End of record (RETURN)	1	200	200	chr\$(13)

FIGURE 10-1 Mailing System Data Record Layout.

<i>Description</i>	<i>Size</i>	<i>From</i>	<i>To</i>	<i>Type</i>
Status character	1	1	1	alpha
File name	10	2	11	alpha
Drive	1	12	12	numeric
Mailing-list title	21	13	33	alpha
Maximum number records	3	34	36	numeric
Names on file	3	37	39	numeric
Today's date	8	40	47	alpha
Last date update	8	48	55	alpha
Unused	146	56	199	alpha
End of record (RETURN)	1	200	200	chr\$(13)

FIGURE 10-2 Mailing System Control Layout.

The *status code* will have four values:

- A—Active record contains data
- C—Control record—first record in file
- D—Deleted record
- U—Unused record

The *date added* will be obtained from the control record and placed here when the record becomes active.

Identification codes will allow you to use up to eight letters to identify the person. Use your imagination to decide how you want to classify and identify a name and address. You might mark all relatives with the letter R and personal friends with the letter F.

The *comments* field is another catchall field but doesn't print on the mailing label. For the newlyweds you may want to enter the date they were married. For the plumber, what he did for you.

The *expansion* field leaves room for you to add new fields that you want in your own personal mailing system. Remember, what the *Companion* demonstrates here is only a model. The advantage of writing your own system is that you can tailor it to your particular needs and desires.

Cassette Tape Works Like A Disk

Your ADAM comes with a single data-pack cassette tape drive that is used as its permanent storage unit. As optional equipment, Coleco offers a disk drive that you can connect to ADAM. A disk drive works like a phonograph record turntable, but instead of reading from the grooves in the record, it reads the magnetic dots on the disk. It has an

arm that can quickly traverse the spinning disk, moving to any of the concentric circles of data blocks on the platter. The arm movement and spinning allow rapid positioning or access to any data block. Audio tape drives used by most home computers read data slowly in one direction. ADAM's digital data drives use high-speed search, forward and backward movement, and high-density digital data-recording techniques to simulate the operation of a disk drive. So, while tape drives and disk drives physically operate differently, in terms of how ADAM accesses data they work the same, except the tape cassette drive works more slowly. Both the tape and disk are formatted in physical blocks of 1024 characters each. Like memory, each block is numbered. You don't have to worry about referring to the blocks by number. ADAM keeps a directory on each tape containing the name of each file and the block where it begins.

While the digital data-cassette tape simulates a disk drive, some of the things that work quickly on the disk take much longer on the cassette and should be avoided. One of them is creating a file randomly. As each additional record is added to the file, ADAM makes a copy of the whole file. Creating a 100-record file using a record number from 0 to 99, rather than sequentially one record after another, could easily take an hour as ADAM copies the file 99 times. The *Companion* recommends creating the file randomly to the expected maximum file size by writing the last record first. So you can read or write randomly any record on the file, every record you write must have the same *fixed* length. You define this length using the L parameter of the OPEN command (see Appendix B). To every record it writes, BASIC adds the end-of-record character (the same character you added to the text line when you pressed the RETURN key). When computing the record length, remember to include the end-of-line character in your character count. The *Little Black Book* demonstrates this technique.

Designing The Screens

Once you know the main processes you want to perform and the data you want to process, you're ready to design the prompt screens that each program displays to tell the operator what information the program needs for it to do its job.

In the *Companion's* Appendix F you will find a text screen layout

form. The *Companion* used this form to lay out the screen for the mailing-label program. Make photocopies of it to design your own screens.

Here are some guidelines for screen design.

- Title each screen so the documentation can identify it and help the operator know what must be done. The *Companion* recommends that you use the top few lines for the title.

- Label all fields the operator must enter. If there is room, show the possible choices or format. For example, to show a yes/no possibility put Y/N after the prompt field.

- Choose 1 or 2 lines for error messages and use the same lines for all programs. This way, when the operator hears the beep meaning an error was made, he or she will know where to look.

- When possible, use the function keys for operator response. Typing one function key is easier and quicker than typing a letter and pressing the return key.

- Tell the operator how to escape from the screen if execution of the program is not to continue.

Having A Program Run Another

The MAILMENU program displays the screen shown in Figure 10-3 (next page). It shows the processes that the mailing-label system will perform. Because it can destroy an existing file, the formatting process has purposely been left off the menu. Just as in SmartWriter, the operator uses the roman-numeral function keys from II to VI to select the process and program to run. Instead of being on the bottom line, the process descriptions are listed in the middle of the screen to give fuller descriptions.

Figure 10-4 (next page) shows the BASIC program. It uses the same GET-key technique as the PICMAKER and LETTRCHASE programs. By placing the RUN command within the MAILMENU program you can execute any of the other mailing-system programs by simply pressing the proper function key. You pass the the RUN command to BASIC in the same way as you pass cassette and disk commands: a PRINT statement with a CHR\$(4) followed by the command string. By convention, you should assign the string variable DS the value CHR\$(4). The *Companion* uses DS instead of CHR\$(4) because DS takes less typing and less room on the line.

TEXT MODE SCREEN

COLUMNS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
1																																1
2																																2
3																																3
4																																4
5																																5
6																																6
7																																7
8																																8
9																																9
10																																10
11																																11
12																																12
13																																13
14																																14
15																																15
16																																16
17																																17
18																																18
19																																19
20																																20
21																																21
22																																22
23																																23
24																																24
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	

LINES

FIGURE 10-3 MAILMENU Screen Layout.


```

10 REM      mailmenu 01/15/84
1000 d$ = CHR$(4): beep$ = CHR$(7)+CHR$(7)
1100 GOSUB 8600: REM      display menu
1200 GET key$
1210 NORMAL
1250 key% = ASC(key$)
1300 IF key% = 3 THEN END
1400 IF key% < 130 OR key% > 134 THEN 2000
1450 PRINT: REM      end any print line
1500 ON key%-129 GOTO 3000, 4000, 5000, 6000, 1700
1700 HOME: PRINT "end of mailmenu"
1720 END
2000 REM      error message
2100 HTAB 1: VTAB 22: PRINT "you pressed invalid key"; beep$;
2200 GOTO 1200
3000 PRINT d$: "run mailmaint"
4000 PRINT d$: "run maillabel"
5000 PRINT d$: "run maillist"
6000 PRINT d$: "run mailctrl"
8600 REM      display mail menu
8605 TEXT
8610 HTAB 8: VTAB 3: PRINT "LITTLE BLACK BOOK";
8620 HTAB 3: VTAB 5: PRINT "---- FUNCTION -----";
8630 HTAB 3: VTAB 7: PRINT "KEY   DESCRIPTION";
8640 HTAB 3: VTAB 9: PRINT "II    FILE MAINTENANCE";
8650 HTAB 3: VTAB 11: PRINT "III   PRINT LABELS";
8660 HTAB 3: VTAB 13: PRINT "IV    PRINT LISTING";
8670 HTAB 3: VTAB 15: PRINT "V     CHANGE CONTROL RECORD";
8680 HTAB 3: VTAB 17: PRINT "VI    END";
8700 HTAB 1: VTAB 21: INVERSE: PRINT "PRESS DESIRED FUNCTION KEY";
8790 RETURN

```

FIGURE 10-4 MAILMENU Program Listing.

Testing The Menu Program

You can easily test this program by creating four short programs, one for each of the programs initiated by the menu program. Type the following program. Use the BASIC line editor to modify statements 10 and 1300 to the proper program name. Then SAVE the program four times, each time using one of the four program names shown on lines 3000 to 6000 in MAILMENU.

```

10 REM      maildummy
400 d$ = CHR$(4)
500 HOME
1300 HTAB 1: VTAB 12: PRINT "maildummy"
1310 HTAB 1: VTAB 14: PRINT "press any key to return to menu"
1320 GET key$
1330 IF key$ = CHR$(3) THEN END
9000 PRINT d$: "run mailmenu"

```

FIGURE 10-5 Dummy Program.

RUN MAILMENU, press a function key and listen to the tape drive load in the selected program. Watch the screen display the program name. Rather than terminate with the END command, each dummy program RUNs MAILMENU. This allows the operator to select the next

program to run. Test that you can run each dummy program from the MAILMENU program.

Now you have the bare bones of the mailing-label system working. Just as the dummy subroutines in PICMAKER let you test that each function key had a subroutine, executing the RUN commands from inside the program lets you test that you could execute each program from MAILMENU and then return to MAILMENU.

String Concatenation

BASIC allows you to join or concatenate strings together in a string expression using a plus sign (+). The mailing-system program uses this technique to build its records, each a single-string variable containing 200 characters. To see how concatenation works, join the word strings "FIRST" and "LAST" together by printing the following string expression.

```
You type      ?"FIRST"+"LAST"  
ADAM displays FIRSTLAST
```

BASIC joined the two words, string constants, together forming a single string. Let's try another example, this time using string variables.

```
You type      first$="ABC"  
You type      last$="xyz"  
You type      together$=first$+last$  
You type      print together$  
ADAM displays ABCxyz
```

The plus sign (+) between the two strings, *first\$* and *last\$*, means you _____ them to create a new string, which you store in the string variable *together\$*. You *join* or *concatenate* strings together using the plus sign symbol (+).

Formatting The File

The formatting program needs some information before it can write and format a random file. The prompt screen shown in Figure 10-6 asks for the necessary information. Even if you don't have a second drive, leave in the prompt for drive. Once you get the mailing-label system working, you may want the convenience and storage capacity

		COLUMNS																																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
LINES	1																																1	
	2																																2	
	3	FORMAT A																															3	
	4																																4	
	5	MAILING FILE																															5	
	6																																6	
	7																																7	
	8																																8	
	9																																9	
	10	ENTER FILENAME X ← → X																															10	
	11																																11	
	12	WHICH DRIVE (1/2)? X																															12	
	13																																13	
	14	HOW MANY RECORDS? XX																															14	
	15																																15	
	16	CHANGED YOUR MIND (Y/N)? X																															16	
	17																																17	
	18																																18	
	19	WAIT! Formatting file																															19	
	20																																20	
	21	X ← → X																															21	
	22																																22	
	23																																23	
	24																																24	

FIGURE 10-6 MAILFORMAT Screen Layout.

that a second tape drive offers, and your program will be ready to use it.

Figure 10-7 (next page) lists the MAILFORMAT program. The program creates a random file from 10 to 99 records depending on the file size entered by the operator. It stores the file size in the proper location in the control record. If you want, you can change the program to allow a larger maximum file size. Remember that with 200-character records and 200,000-character storage space on the


```

100 REM mailformat 1/28/84
1000 GOSUB 6000: REM      initialization
1100 GOSUB 5000: REM      display prompt screen
1200 GOSUB 2000: REM      input data
1400 IF format% THEN GOSUB 3000
1500 END
2000 REM      input data
2100 HTAB 19: VTAB 10: INPUT "": file$
2200 IF LEN(file$) >= 0 OR LEN(file$) <= 10 THEN 2300
2220 err% = 1: GOSUB 7000: GOTO 2100
2300 HTAB 19: VTAB 12: INPUT "": drive$
2320 IF drive$ = "1" OR drive$ = "2" THEN 2400
2340 err% = 2: GOSUB 7000: GOTO 2300
2400 HTAB 19: VTAB 14: INPUT "": size$
2410 IF LEN(size$) <> 1 THEN 2450
2420 IF size$ >= "1" AND size$ <= "9" THEN 2490
2450 IF LEN(size$) <> 2 THEN 2480
2460 IF size$ >= "10" AND size$ <= "99" THEN 2490
2480 err% = 3: GOSUB 7000: GOTO 2400
2490 size = VAL(size$)
2600 HTAB 26: VTAB 16: INPUT "": ans$
2620 IF ans$ = "n" OR ans$ = "N" THEN format% = true%
2900 control$ = "C"+LEFT$(file$+LEFT$(sp$, 10), 10)+drive$
2910 control$ = control$+LEFT$(sp$, 21)+RIGHT$("000"+size$, 3)+"000"
2920 control$ = control$+LEFT$(sp$, 160)
2990 RETURN
3000 REM      format file
3010 HTAB 7: VTAB 19: PRINT "WAIT! Formatting file"
3050 PRINT d$: "open ": file$: ",L200,D": drive$
3100 FOR record = size TO 1 STEP -1
3150 PRINT d$: "write ": file$: ",R": record
3200 record$ = "URECORD "+RIGHT$( " "+STR$(record), 2)+LEFT$(sp$, 189)
3300 PRINT record$
3600 NEXT record
3670 PRINT d$: "write ": file$: ",R0"
3680 PRINT control$
3700 PRINT d$: "close ": file$
4900 RETURN
5000 REM      display prompts
5100 ROME
5200 HTAB 12: VTAB 3: PRINT "FORMAT A"
5300 HTAB 10: VTAB 5: PRINT "MAILING FILE"
5400 HTAB 4: VTAB 10: PRINT "ENTER FILENAME":
5500 HTAB 1: VTAB 12: PRINT "WHICH DRIVE(1/2)?"
5600 HTAB 1: VTAB 14: PRINT "HOW MANY RECORDS?"
5700 HTAB 1: VTAB 16: PRINT "CHANGED YOUR MIND(Y/N)?":
5900 RETURN
6000 REM      initialize values
6050 d$ = CHR$(4): beep$ = CHR$(7)+CHR$(7)
6070 true% = 1: false% = 0
6100 sp$ = " ": FOR i = 1 TO 7: sp$ = sp$+sp$: NEXT i
6110 sp$ = sp$+LEFT$(sp$, 127): REM      255 spaces
6900 RETURN
7000 REM      errors
7100 HTAB 1: VTAB 21
7200 IF err% = 1 THEN PRINT "name too short/long"
7300 IF err% = 2 THEN PRINT "ONLY 1 OR 2!":
7400 IF err% = 3 THEN PRINT "only 1 to 99 size allowed"
7900 PRINT beep$:
7990 RETURN

```

FIGURE 10-7 MAILFORMAT Program Listing.

tape after allowing space for the directory and BASIC programs, the tape has a 1000-name capacity.

After typing the program, SAVE it and then RUN it. You can use SmartWriter in the window display mode to take a look at the records created. Just use the down arrow to scroll down. Don't STORE the file back on tape. Remember, SmartWriter is a word processor, not a simple text editor. Built into SmartWriter is reformatting logic to

make the text look neat. It really doesn't expect to see a record like the MAILFORMAT program created.

After you have tested the MAILFORMAT and MAILCONTRL programs and have confidence that they both work, you may want to change statement 1990 in MAILFORMAT so that it RUNs MAILCONTRL so you can update a control record immediately after formatting the file.

FORMATTED FIELD PROCESSOR

A field is a group of related characters, like the name of a file or program. A field can contain constant characters, like ENTER FILENAME=, or, like a variable, a field may contain characters that change. Usually the term *field* refers to a variable field. Fields occur on the screen, in records and in printed reports. A variable is a field in memory. On the screen and report layouts, a field is indicated by X's connected by dashes to indicate the size of the field, like X---X to represent a five-character field. A description ending with dashes and X also represents a field that changes. For example, ERROR MESSAGE-----X represents a field used to display a 20-character error message.

When the operator enters the eleventh character, you want the program to beep and display an error message. To do this you need an input routine that uses the GET command to get one character at a time from the keyboard.

By now you should be familiar with the idea of programs that process data to perform particular processes or functions. SmartWriter, a word processor, allows you to input, change, print, store and retrieve documents consisting of free-form words grouped in text lines. The PICMAKER program allows you to process colored squares to draw pictures. To input the change data displayed as fixed-length fields you need a formatted field processor, which the *Companion* calls *fieldinput*.

Think about the functions you will need in the field processor. SmartWriter and PICMAKER used the arrow.

When entering data you should limit the number of characters entered to the length of the field, because that's all the room you have for it in the record. If the operator tries to enter more characters, the program should display an error message, so the operator's aware of making a mistake. The INPUT command limits the size of a string input field to 255 characters. You can't set a limit to just 10 characters.

In the mailing-system data-entry programs you want to limit the number of characters the operator can enter. For example, ADAM limits file names to 10 keys to make the cursor move around the screen. You can use the left and right arrows to move back and forth within a field. The up and down arrows will allow you to move up and down from field to field. You can use the backspace key to move the cursor left, deleting the preceding character.

To tell the program you're done processing, you can press one of the function keys. SmartWriter often uses VI as the DONE key and so will *fieldinput*. To QUIT without updating, the operator will press the V function key.

The *fieldinput* routine must process any field on the screen. This requires that parameters or values define the types of information, called attributes, of each field. Each field has the following common attributes:

- Row for prompt and data
- Beginning column for prompt
- Beginning column for data
- Length of data field
- Position of data in record
- Type of field—alphanumeric or numbers only
- Prompt—for example, ENTER FILENAME

Given this information, the *field%* array *fieldinput* (diagrammed in Figure 10-8) can move the cursor to the correct position on the screen for the operator to enter data, make sure the operator does not enter more characters than the field length, check the character entered for validity and store it in the data record. Using SmartWriter, you can merge the *fieldinput* routine into any program where you want to control the operator's data entry.

Figure 10-9 (p. 240) lists the *fieldinput* routine. Enter it and SAVE as a separate program. You can't RUN it because it needs input from the program using it. LIST the routine and do a line-by-line check to make sure you made no typing mistakes.

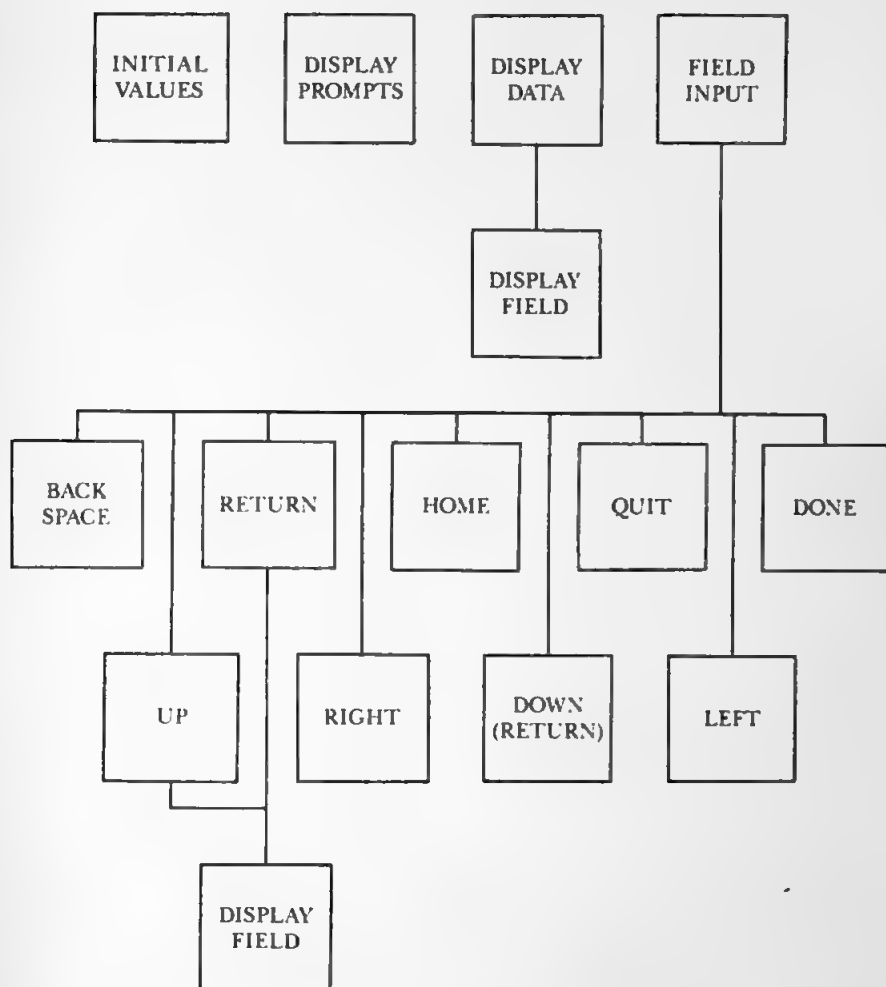


FIGURE 10-8 Diagram of FIELDINPUT.


```

15100 REM FIELDINPUT 01/28/84
15105 exit% = false%: quit% = false%
15110 GOSUB 15430: REM init field
15120 HTAB ic%: VTAB rt%: GET key$: key% = ASC(key$)
15125 IF key% = 3 THEN NORMAL: END
15130 HTAB l%: VTAB 21: NORMAL: PRINT LEFT$(sp$, 31);
15140 HTAB ic%: VTAB rt%: INVERSE
15150 IF key% >= 32 AND key% < 128 THEN 15210
15160 FOR func = 1 TO 9
15165 IF func%(func) = key% THEN 15170
15166 NEXT func
15167 err% = 3: GOSUB 15800: GOTO 15120
15170 ON func GOSUB 15350, 15400, 15440, 15470, 15480, 15600, 15650, 15400, 1570
0
15175 REM bs,return,home,quit,done,up,right,down,left
15177 IF exit% THEN NORMAL: RETURN
15180 GOTO 15120
15210 IF ic% = c%+1% THEN err% = 1: GOSUB 15800: GOTO 15120
15220 ON t% GOSUB 15300, 15320
15230 GOTO 15120
15300 IF key$ < "0" OR key$ > "9" THEN err% = 2: GOSUB 15800: RETURN
15320 PRINT key$: : ic% = ic%+1
15330 record%(ip%) = key$: ip% = ip%+1: RETURN
15350 IF ic% = c% THEN PRINT beep$: : RETURN
15360 ic% = ic%-1: HTAB ic%: PRINT " ";
15370 ip% = ip%-1: record%(ip%) = ASC(" "): RETURN
15400 NORMAL: HTAB ct%: VTAB rt%: GOSUB 15450
15405 IF field% = t% THEN field% = f%: GOTO 15430
15410 field% = field%+1
15430 c% = field%(field%, col%)
15432 r% = field%(field%, row%)
15434 l% = field%(field%, size%)
15436 p% = field%(field%, post%)
15438 m% = field%(field%, type%)
15440 HTAB ct%: VTAB rt%: INVERSE: GOSUB 15450
15444 ic% = ct: ip% = p%
15449 RETURN
15450 REM display field
15460 FOR ip = p% TO p%+l%-1: PRINT CHR$(record%(ip)): : NEXT ip: RETURN
15470 quit% = true%
15480 exit% = true%: RETURN
15500 REM display fields
15510 FOR field = f% TO t%
15520 c% = field%(field, col%)
15522 r% = field%(field, row%)
15524 l% = field%(field, size%)
15526 p% = field%(field, post%)
15530 NORMAL: HTAB ct%: VTAB rt%: GOSUB 15450
15540 NEXT field
15550 RETURN
15600 NORMAL: HTAB ct%: VTAB rt%: GOSUB 15450
15605 IF field% = f% THEN field% = t%: GOTO 15430
15610 field% = field%-1: GOTO 15430
15650 REM right
15660 IF ic% = c%+1% THEN err% = 1: GOSUB 15800: RETURN
15670 ic% = ic%+1: ip% = ip%+1: RETURN
15700 IF ic% = c% THEN PRINT beep$: : RETURN
15710 ic% = ic%-1: ip% = ip%-1: RETURN
15800 REM input error
15810 HTAB l%: VTAB 21: PRINT msg$(err%): beep$: RETURN
15820 REM INITIALIZE FIELDINPUT
15825 msg$(1) = "TOO LONG! TYPE return or bs"
15830 msg$(2) = "NOT NUMERIC!"
15835 msg$(3) = "INVALID KEY!"
15840 DATA 8,13,128,133,134
15845 REM bs,return,home,quit(v),done(vi)
15850 DATA 160,161,162,163
15855 REM up,right,down,left
15860 FOR func = 1 TO 9

```



```

15865 READ keyvalue: func%(func) = keyvalue
15870 NEXT func
15874 space% = ASC(" "): REM          space
15875 FOR p = 1 TO 200: record%(p) = space%: NEXT p
15880 TEXT
15885 HTAB 21: VTAB 23: PRINT "  V    VI";
15890 HTAB 21: VTAB 24: PRINT "QUIT DONE";
15899 RETURN
15900 REM  DISPLAY PROMPTS
15910 FOR field = ff% TO tf%
15920 HTAB field%(field, pcol%): VTAB field%(field, row%)
15930 PRINT prompt$(field)
15940 NEXT field
15950 RETURN

```

FIGURE 10-9 FIELDINPUT Listing.

Using Fieldinput

The *fieldinput* routine supports the following function keys:

BACKSPACE Blank out the previous character and move the cursor left one position.

RETURN Move cursor down to next field. From the bottom field it moves to the first or top field.

HOME Move cursor to beginning of current field.

FUNCTION V QUIT the *fieldinput* routine with *quit%* true. The program uses the *quit%* switch to test if the operator doesn't want to make the changes entered. If *quit%* is true, the program doesn't write the record back to the tape.

FUNCTION VI DONE, exit *fieldinput*. The operator is done entering data or making changes.

UP ARROW Move cursor up to beginning of previous field. If at the first field, move to last field.

→ Move cursor right. Limited to end of field.

DOWN Identical to RETURN key. Move cursor down.

← Move cursor left. Limited to beginning of field.

The program must pass *fieldinput* information in arrays and variables.

record% Stores each character in the record as an integer to make it easy for *fieldinput* to access and change a character in the record.

After reading the record, the program must convert the string record into the *record%* integer array using a FOR . . . NEXT loop similar to:

```
9370 FOR p=1 TO 200: record%(p)=ASC(MID$(control$,
p,1)): NEXT p
```

The MID\$ function selects a character from *control\$*, and the ASC function converts the character to an integer.

You must DIMension *record%* to the record size. For example, the mailing system has a 200-character record so make the DIMension

```
100 DIM record%(200)
```

func% Stores the values of the nine function keys. Always dimension as *func%(9)* because you always use all the keys.

prompt\$ A string array used to store the prompts for each input.

Dimension this array to the number of fields. For example, the program MAILCONTRL has seven fields so make the DIM *prompt\$(7)*.

field% A two-dimensional array that stores the five attributes of each field. The first subscript defines the number of fields and the second subscript the six attributes.

Instead of using meaningless numbers, *fieldinput* assigns the six subscript values for the attributes to integer variables. The table below shows the name, subscript value and description for the six attributes.

Variable Name	Subscript Value	Description
<i>row%</i>	1	Row for prompt and data
<i>pcol%</i>	2	Column for prompt
<i>col%</i>	3	Beginning column for data field
<i>size%</i>	4	Length of data field
<i>pos%</i>	5	Beginning position of data field in record. Used as subscript to <i>record%</i> array to retrieve and change data.
<i>type%</i>	6	Type data: 1=number, 2=any character

For example, the MAILCONTRL screen in Figure 10-10 shows seven input fields so you use seven for the first subscript. Dimension the

TEXT MODE SCREEN

		COLUMNS																																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
LINES	1																																		1
	2																																		2
	3																																		3
	4																																		4
	5																																		5
	6																																		6
	7																																		7
	8																																		8
	9																																		9
	10																																		10
	11																																		11
	12																																		12
	13																																		13
	14																																		14
	15																																		15
	16																																		16
	17																																		17
	18																																		18
	19																																		19
	20																																		20
	21																																		21
	22																																		22
	23																																		23
	24																																		24

FIGURE 10-10 MAILCONTRL Screen.

field% array as DIM(7,5). The third field, maximum number of records in file, has the prompt MAXIMUM # RECORDS, which appears on row 9; the prompt begins in column 1 and the data begins in column 18. The data field has three characters, and each character gets stored in the record array as an integer beginning at the 34th element. Move these values into *field%* using the proper subscripts.

For example, to assign the row attribute for the third field write:
`field%(3,row%)=9`

The variables passed are subscripts for *field%* array. They indicate which fields on the screen to process.

<i>Variable</i>	<i>Description</i>
<i>tf%</i>	To field or last field
<i>ff%</i>	From field or first field
<i>field%</i>	Current field

For example, to change data in fields 4 through 7 and position the cursor at the fifth, make the following assignments:

`1000 ff%=4:tf%=7:field%=5`

Programming Using Fieldinput

To use *fieldinput* the program must:

1. Define the arrays as already described.
2. GOSUB 15820 to initialize the *func%*, error message array and display the QUIT and DONE prompts at the bottom of the screen.
3. Load the *prompt\$* and *field%* arrays with field information.
4. Display the prompt screen.
5. Convert the record string you want to display and change into the integer array *record%*.
6. Display the existing values by assigning *ff%* to the first field and *tf%* to the last field and then doing a GOSUB 15500 to display the data fields between the FROM and TO fields, *ff%* and *tf%* inclusive.
7. Assign values to *tf%* and *ff%* to define the range of field you want to change.
8. Assign a value to *field%* to position the cursor at the first field you want the operator to review for possible change.
9. Do a GOSUB 15100 to *fieldinput* that allows the operator to make changes to the data-record integer array.
10. Check the *quit%* switch.
11. If true, don't write the record to the file.
12. If false, convert the integer array *record%* back to a string record and write the record to the file on the cassette tape.

The MAILCONTRL program (Figure 10-11) demonstrates this procedure.


```

10 REM mailcontrol 1/28/84
100 DIM record$(200), func$(9), field$(7, 6), prompt$(7)
1000 REM mail process
1100 GOSUB 8500: REM INITIALIZATION
1200 GOSUB 15820: REM init fieldinput
1220 GOSUB 8900: REM DISPLAY PROMPTS
1240 f% = 1: t% = 2: field% = f%
1250 GOSUB 15100: REM FIELDINPUT
1260 IF quit% THEN 1900
1265 file$ = ""
1270 FOR j = 2 TO 11: char% = record$(j): IF char% = space% THEN char% = 0
1275 file$ = file$+CHR$(char%): NEXT j
1280 IF LEN(file$) = 0 THEN GOTO 1910
1290 drive$ = CHR$(record$(12))
1295 IF drive$ = "1" OR drive$ = "2" THEN 1310
1300 HTAB 1: VTAB 22: PRINT "INVALID DRIVE": GOTO 1240
1310 GOSUB 9000: REM READ CONTOL
1320 IF found% THEN 1700
1330 HTAB 1: VTAB 22: PRINT "FILE ": file$: " NOT FOUND": beep$:
1340 GOTO 1240
1700 REM read and process
1740 f% = 3: t% = 7: GOSUB 15900: REM PROMPTS
1750 GOSUB 15500: REM DISPLAY DATA
1760 field% = 4: f% = 4: GOSUB 15100: REM FIELDINPUT
1770 IF quit% THEN 1900
1810 GOSUB 9400: REM write control
1900 REM end of program
1910 HOME: PRINT "end of mailcontrol"
1990 END
8500 $ = CHR$(4): beep$ = CHR$(7)+CHR$(7)
8520 row% = 1: pcol% = 2: col% = 3: size% = 4: pos% = 5: type% = 6
8530 true% = 1
8540 sp$ = " ": FOR i = 1 TO 6: sp$ = sp$+sp$: NEXT i
8600 REM INITIALIZE FIELDINPUT ARRAYS
8630 FOR field = 1 TO 7
8640 READ field$
8650 field% = field: REM integer subscripts execute faster
8680 prompt$(field%) = RIGHT$(field$, LEN(field$)-12)
8690 field$(field%, row%) = VAL(LEFT$(field$, 2))
8700 field$(field%, pcol%) = VAL(MID$(field$, 3, 2))
8710 field$(field%, col%) = VAL(MID$(field$, 5, 2))
8720 field$(field%, size%) = VAL(MID$(field$, 7, 2))
8730 field$(field%, pos%) = VAL(MID$(field$, 9, 3))
8740 field$(field%, type%) = VAL(MID$(field$, 12, 1))
8750 NEXT field
8790 RETURN
8800 REM
8810 REM field table
8810 REM row,prompt column,data column,field length, pos
8810 REM
8820 REM --RwPcdClnPostPprompt
8840 DATA "050118100022ENTER FILE NAME"
8850 DATA "070118010121ENTER DRIVE(1/2)"
8860 DATA "090118030342MAX # RECORDS"
8870 DATA "110111210132LIST NAME"
8880 DATA "130118030371NAMES ON FILE"
8885 DATA "150118080402TODAY'S DATE"
8890 DATA "170118080482LAST UPDATE"
8900 REM DISPLAY PROMPT SCREEN
8910 HOME: HTAB 3: PRINT "CONTROL RECORD MAINTENANCE"
8920 f% = 1: t% = 2: GOSUB 15900
8930 GOSUB 15885: REM done and quit prompts
8990 RETURN
9000 REM read control record
9190 ONERR GOTO 9700
9195 HTAB 1: VTAB 21: PRINT d$
9200 PRINT d$: "open ": file$: ",L200,D": drive$
9300 PRINT d$: "read ": file$: ",R0"
9350 INPUT "": control$
9355 CLRERR: found% = true%
9360 PRINT d$
9365 cln = LEN(control$)
9370 FOR p = 13 TO cln: record$(p) = ASC(MID$(control$, p, 1)): NEXT p
9390 RETURN
9400 REM re-write control
9404 control$ = "c"
9406 FOR p = 2 TO 199: control$ = control$+CHR$(record$(p)): NEXT p
9500 HTAB 1: VTAB 21: PRINT d$

```



```
9510 PRINT d$: "write "; file$: ",R0"  
9600 PRINT control$  
9650 PRINT d$: "close "; file$  
9690 RETURN  
9700 REM          onerr  
9710 CLRERR: IF ERRNUM(0) = 5 THEN 9800  
9720 PRINT "ERRNUM="; ERRNUM(0)  
9730 RETURN  
9800 REM          file not found  
9810 PRINT d$: "close "; file$  
9820 PRINT d$: "delete "; file$  
9830 found% = false  
9990 RETURN
```

FIGURE 10-11 MAILCONTRL Listing.

UPDATING AND DISPLAYING THE CONTROL RECORD

The control record is the first record on the mailing file. You want to update this record for the current date every time you use the system. After formatting the file you want to enter the mailing-list name that the other programs display on the top line of their screens. The MAILMAINT program will update the records on the file. You may want to look at the file after an update session so you know how much room you have left.

Random Access Of Records

To read or write a file randomly by record number, rather than sequentially from beginning to end, you need to give BASIC additional information.

On the open statement, immediately after the filename, you define the record length. All records in the file must have the same number of characters, and the length must include the return character appended by BASIC. For example, suppose you use MAILFORMAT to create a file called *friends*. To read the first record from the file, which has record length of 200 characters, from tape drive 1 you must write the following statements:

```
9200 PRINT d$; "OPEN FRIENDS,L201,D1"  
9300 PRINT d$; "READ FRIENDS,R0"  
9340 CLRERR  
9350 INPUT " ";control$  
9360 PRINT d$
```

It's assumed *d\$* has the value CHR\$(4). The letter L after the filename indicates that the number 201 defines the record length. One was added to 200 to account for the RETURN character, which is stored on the tape but not in the variable *control\$*. The letter D indicates that the one is the drive number; the left drive is one and the right is two. You must separate the different parameters of the OPEN statement with commas.

The READ command tells BASIC to get the next INPUT from the tape file. The letter R following the comma indicates the value, in this case zero, of the record number you want to read. The first record on the file has a record number zero. The MAILMAINT program requires you to enter the record number of the person whose information you want to change, delete or display.

If you successfully get to statement 9340 you have read a record, and CLRERR turns off the ONERR command so that regular BASIC error handling occurs when BASIC detects an error. The special ONERR handling for a filename not found in the tape directory is explained in the next section.

The INPUT statement reads the record from the tape cassette into the string variable *control\$*.

The PRINT *d\$* statement tells BASIC that the next INPUT will come from the keyboard and *not* the tape.

After using *fieldinput* to update a record, like the *control\$* record, you want to write it back onto the tape in the same place. You use the WRITE command with a record number. For example,

```
9500 PRINT d$;"WRITE FRIENDS,R0"
```

```
9600 PRINT control$
```

```
9650 PRINT D$;"CLOSE FRIENDS"
```

The letter R following the comma indicates that the value, in this case zero, is the record number location where you want to write the record.

The PRINT *control\$* statement writes the record to tape.

When the program finishes reading and writing to a file, you must tell BASIC to CLOSE the file. Failure to close the file leads to unpredictable results. Should the program terminate without closing the file, you must type, for example, the following.

```
?CHR$(4);"CLOSE FRIENDS"
```

This closes the file, changing the blocks-used value in the directory to the actual size from the number of unused blocks on the tape set when BASIC opened the file.

To allow MAILCONTRL to process different filenames, the program uses a string variable for the filename rather than the string constant FRIENDS, but performs the same processes as just described.

Remember, to read or write a record randomly, you need two statements. To READ you must use a PRINT d\$ with the READ command and record number followed by an INPUT statement. To WRITE you must use a PRINT d\$ with a WRITE command and record number followed by a PRINT statement. The string variable d\$ must be assigned the value CHR\$(4).

File Doesn't Exist

If the filename doesn't match a name in the tape directory, BASIC normally STOPS execution of the program. To prevent this from happening so you have another chance to enter the right filename, you use the ONERR statement.

Just before the first READ statement place a statement similar to the following:

```
9190 ONERR GOTO 9700
```

At line number 9700, in our example, you test the ERRNUM (0)

function for a 5, which means "end of data." The OPEN statement automatically creates a file when it can't find the name in the directory. When you READ it, the program finds it empty—causing the end-of-data error. You want to close and delete the erroneously created file. Deleting the incorrectly spelled filename frees space on the tape and in the directory.

For example, if *friends* didn't exist the ONERR routine would look like:

```
9710 IF errnum (0) = 5 THEN 9800
9720 PRINT "ERRNUM=";errnum(0)
9730 STOP
9800 CLRERR
9810 PRINT d$;"CLOSE FRIENDS"
9820 PRINT d$;"DELETE FRIENDS"
9840 GOTO 1000
```

The GOTO goes to the start of the main program loop, starting the program over and allowing the operator to reenter the correct filename. Your programs may use different line numbers but use the same commands to handle a filename not found.

DATA MAINTENANCE

The MAILMAINT program gives you the ability to add, change, delete and display the data on the mailing file. Figure 10-13 (next page) shows the top-down structure. The major process within the main process consists of displaying a menu so the operator can choose the maintenance operation to perform. Figure 10-12 (next page) shows the screen layout. The operator can choose to:

- Add name and address records to the mailing file.
- Change information in a name and address record.
- Delete a name and address record. No actual physical deletion takes place, the program just changes the status field to D. The MAILLABEL program checks the status field and ignores all D and U records.

Figure 10-14 (p. 252) shows the program listing of MAILMAINT except for *fieldinput* shown in Figure 10-9. Use SmartWriter to combine the two listings to make one program.

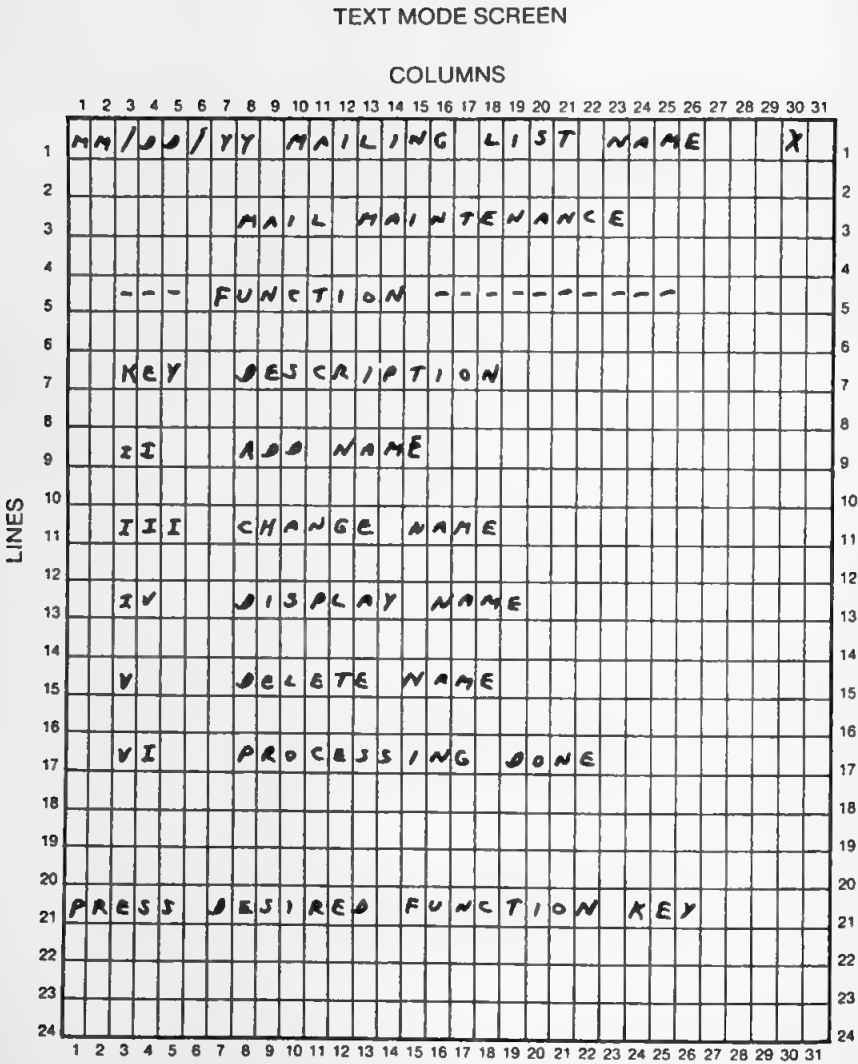


FIGURE 10-12 Mail Maintenance Menu—Screen Layout.

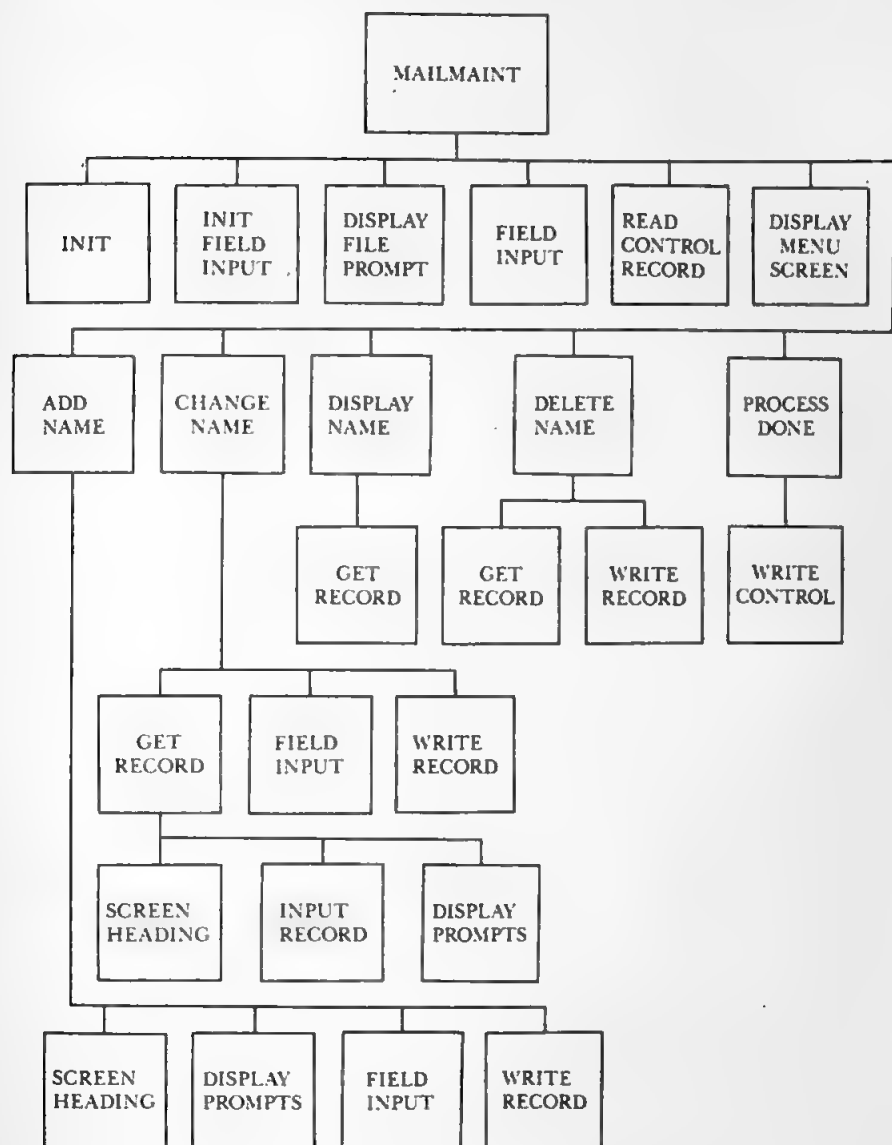


FIGURE 10-13 Diagram of MAILMAINT.


```

10 REM mailmaint 01/20/84
100 DIM record%(200), func%(9), field%(15, 6), prompt$(15)
1000 REM get file name and read control
1100 GOSUB 8500: REM initialization
1200 GOSUB 15820: REM init fieldinput
1220 GOSUB 8900: REM display file and drive prompts
1240 f% = 1: t% = 2: field% = f%
1250 GOSUB 15100: REM fieldinput
1260 IF quit% THEN 1900
1265 file$ = ""
1270 FOR j = 2 TO 11: char% = record%(j): IF char% = space% THEN char% = 0
1275 file$ = file$+CHR$(char%): NEXT j
1280 IF LEN(file$) = 0 THEN 1910
1290 drive$ = CHR$(record%(12))
1295 IF drive$ = "1" OR drive$ = "2" THEN 1310
1300 HTAB 1: VTAB 22: PRINT "INVALID DRIVE"; beep$: GOTO 1240
1310 GOSUB 9000: REM read control
1320 IF found% THEN 1700
1330 HTAB 1: VTAB 22: PRINT "FILE "; file$: " NOT FOUND"; beep$
1340 GOTO 1240
1700 REM main process
1710 GOSUB 7000: REM selections screen
1720 GET key$: key% = ASC(key$)
1730 IF key% = 3 THEN END
1740 IF key% >= 130 AND key% <= 134 THEN 1780
1750 HTAB 1: VTAB 22: PRINT "You pressed invalid key"; beep$
1760 GOTO 1720
1780 ON key%-129 GOSUB 2000, 3000, 4000, 5000, 1900
1790 REM add, change, display, delete
1799 GOTO 1700
1900 REM end of program
1930 GOSUB 9400: REM re-write control
1940 HOME: PRINT "end of mailmaint"
1990 END
2000 REM add record
2005 GOSUB 7800: GOSUB 15885
2010 HTAB 1: VTAB 3: PRINT "ADD"
2020 IF last% < max% THEN 2100
2030 HTAB 1: VTAB 22: PRINT "NO MORE ROOM": RETURN
2100 number = last%+1
2105 HTAB 16: VTAB 3: PRINT "NUMBER="; number
2110 f% = 4: t% = 15: GOSUB 15900
2120 HTAB 1: VTAB 7: PRINT "ADDRESS LINES 2-4"
2130 FOR p = 1 TO 199: record%(p) = ASC(" "): NEXT p
2140 field% = 4: GOSUB 15100: REM allow changes
2150 IF quit% THEN RETURN
2155 record%(1) = ASC("A")
2160 GOSUB 3800: REM write
2170 last% = number%
2990 RETURN
3000 REM change record
3010 op$ = "CHANGE": GOSUB 6000
3020 IF quit% THEN RETURN
3030 field% = 4: GOSUB 15100: REM allow changes
3040 IF quit% THEN RETURN
3050 GOSUB 3800: REM write
3590 RETURN
3800 REM write record
3805 HTAB 1: VTAB 21: PRINT d$
3810 PRINT d$: "write "; file$: ",R"; number
3820 record$ = ""
3830 FOR p = 1 TO 200: record$ = record$+CHR$(record%(p)): NEXT p
3840 PRINT record$
3850 HTAB 1: VTAB 21: PRINT d$
3890 RETURN
3990 RETURN
4000 REM display
4010 op$ = "DISPLAY": GOSUB 6000
4020 GET key$: key% = ASC(key$)
4030 IF key% <> 134 THEN 4020
4990 RETURN
5000 REM delete
5010 op$ = "DELETE": GOSUB 6000
5020 GET key$: key% = ASC(key$)
5030 IF key% = 133 THEN RETURN: REM quit
5040 IF key% <> 134 THEN 5020
5050 record%(1) = ASC("D")
5060 GOSUB 3800
5990 RETURN

```



```

6000 REM          get number
6100 GOSUB 7800: GOSUB 15885
6110 HTAB 1: VTAB 3: PRINT op$:
6120 fff% = 3: tff% = 3: GOSUB 15900
6130 field% = fff%: GOSUB 15100
6140 IF quit% THEN RETURN
6150 fff% = 4: tff% = 15: GOSUB 15900
6160 HTAB 1: VTAB 7: PRINT "ADDRESS LINES 2-4"
6200 number$ = ""
6210 FOR p = 1 TO 3: number$ = number$+CHR$(record%(p)): NEXT p
6220 number = VAL(number$)
6230 IF number <> 0 THEN 6300
6250 HTAB 1: VTAB 22: PRINT "INVALID NUMBER": beep$
6260 GOTO 6120
6300 ONERR GOTO 6500
6400 PRINT d$: "read "; file$: ",R": number
6410 INPUT "": record$
6420 HTAB 1: VTAB 21: PRINT d$: CLRERR
6430 FOR p = 1 TO 199: record%(p) = ASC(MID$(record$, p, 1)): NEXT p
6450 GOSUB 15500: REM          display fields
6490 RETURN
6500 CLRERR: quit% = true%
6510 PRINT "RECORD "; number: " NOT FOUND": beep$
6590 RETURN
7000 REM          display selection screen
7120 GOSUB 7800: REM          top line
7140 HTAB 8: VTAB 3: PRINT "MAIL MAINTENANCE":
7150 HTAB 3: VTAB 5: PRINT "---- FUNCTION -----":
7160 HTAB 3: VTAB 7: PRINT "KEY   DESCRIPTION"
7170 HTAB 3: VTAB 9: PRINT "II    ADD NAME":
7180 HTAB 3: VTAB 11: PRINT "III   CHANGE NAME":
7190 HTAB 3: VTAB 13: PRINT "IV    DISPLAY NAME":
7200 HTAB 3: VTAB 15: PRINT "V     DELETE NAME":
7210 HTAB 3: VTAB 17: PRINT "VI    PROCESSING DONE":
7220 HTAB 1: VTAB 21: PRINT "PRESS DESIRED FUNCTION KEY"
7790 RETURN
7800 HOME: PRINT MID$(controls$, 40, 8): " "; MID$(controls$, 13, 21)
7990 RETURN
8500 d$ = CHR$(4): beep$ = CHR$(7)+CHR$(7)
8520 row% = 1: pcol% = 2: col% = 3: size% = 4: pos% = 5: type% = 6
8530 true% = 1
8540 sp$ = " ": FOR i = 1 TO 6: sp$ = sp$+sp$: NEXT i
8600 REM          INITIALIZE FIELDINPUT ARRAYS
8630 FOR field = 1 TO 15
8640 READ field$
8650 field% = field: REM          integer subscripts execute faster
8680 prompt$(field%) = RIGHTS$(field$, LEN(field$)-12)
8690 field%(field%, row%) = VAL(LEFT$(field$, 2))
8700 field%(field%, pcol%) = VAL(MID$(field$, 3, 2))
8710 field%(field%, col%) = VAL(MID$(field$, 5, 2))
8720 field%(field%, size%) = VAL(MID$(field$, 7, 2))
8730 field%(field%, pos%) = VAL(MID$(field$, 9, 3))
8740 field%(field%, type%) = VAL(MID$(field$, 12, 1))
8750 NEXT field
8790 RETURN
8800 REM          field table
8810 REM          row,prompt column,data column,field   length,
          position,type
8820 REM          --RwPcDcLnPostFPrompt
8825 DATA          "050118100022ENTER FILE NAME"
8830 DATA          "070118010121ENTER DRIVE(1/2)"
8835 DATA          "031023030011ENTER NUMBER"
8840 DATA          "050113120022FIRST NAME"
8845 DATA          "060113170142LAST NAME"
8850 DATA          "090101300312 "
8855 DATA          "100101300612 "
8860 DATA          "110101300912 "
8865 DATA          "130112141212CITY"
8870 DATA          "140112021352STATE"
8875 DATA          "150112051371ZIP"
8880 DATA          "160112101421TELEPHONE"
8885 DATA          "170112081522DATE ADDED"
8890 DATA          "180112081602MAIL CODES"
8895 DATA          "190112201682COMMENT"
8900 REM          DISPLAY PROMPT SCREEN
8910 fff% = 1: tff% = 2: GOSUB 15900
8920 HTAB 8: VTAB 3: PRINT "MAIL MAINTENANCE":
8990 RETURN
9000 REM          read control record
9190 ONERR GOTO 9700

```



```

9195 PRINT d$
9200 PRINT d$: "open "; file$: ",L200,D"; drive$
9300 PRINT d$: "read "; file$: ",R0"
9350 INPUT "": control$
9355 CLRERR: found$ = true$
9360 PRINT d$
9370 last$ = VAL(MID$(control$, 37, 3))
9375 max$ = VAL(MID$(control$, 34, 3))
9390 RETURN
9400 REM re-write control
9410 control$ = LEFT$(control$, 36)+RIGHT$("000"+STR$(last$), 3)+RIGHT$(control
$, 160)
9420 control$ = LEFT$(control$, 47)+MID$(control$, 40, 8)+RIGHT$(control$, 145)

9500 PRINT d$
9510 PRINT d$: "write "; file$: ",R0"
9600 PRINT control$
9650 PRINT d$: "close "; file$
9690 RETURN
9700 REM onerr
9710 CLRERR: IF ERRNUM(0) = 5 THEN 9800
9720 PRINT "ERRNUM=": ERRNUM(0)
9730 RETURN
9800 REM file not found
9810 PRINT d$: "close "; file$
9820 PRINT d$: "delete "; file$
9830 found$ = false$
9990 RETURN

```

FIGURE 10-14 MAILMAINT Listing.

Adding Records

To add a record the program does the following:

1. Displays the prompt screen.
2. Clears the data array to spaces because no previous information about this new person exists on the file.
3. Uses the *fieldinput* routine to allow the operator to enter data into each field.
4. Assigns a record number that is one more than the last number used. The record number of the last record used is stored in the control record.
5. If the operator exits *fieldinput* by pressing the QUIT function key, *quit%* is true and the record is not added to the file. The last-used record number is adjusted down by one.
6. If *quit%* is false the program writes the record to the file.
7. Update control record for last record used.

Changing Records

To change a record the program does the following:

1. Displays the prompt screen.
2. Asks the operator to enter a record number.
3. Reads the record and converts it into the data array.
4. Uses *fieldinput* to display the data fields.
5. Uses *fieldinput* to allow the operator to make changes to the data fields.
6. If the operator exits *fieldinput* by pressing the QUIT function, *quit%* is true, and the record is not written to the tape so the information on the tape remains unchanged.
7. If *quit%* is false the program writes the record with the new information to file.

Deleting Records

To delete a record the program does the following:

1. Displays the prompt screen.
2. Asks the operator to enter a record number.
3. Reads the record and converts it into a data array.
4. Displays the data fields.
5. Asks if the operator want to delete this record.

6. If yes, it moves D to status code and writes the record.
7. If no, the program displays the maintenance menu.

Displaying Records

To display a record the program does the following:

1. Displays the prompt screen.
2. Asks the operator to enter a record number.
3. Reads the record and converts it into an integer array.
4. Displays the data fields.
5. Waits for the operator to press the DONE function key.
6. When the operator presses DONE the program displays the maintenance menu.

MAILING LABELS

The mailing-label program gives you the payoff. It automatically types the mailing labels on the printer, making worthwhile all the effort to write the programs and type the names into the maintenance program.

Figure 10-15 shows the screen layout. Like all the mailing-system programs, you must enter the filename and tape drive number. Since the program has only three input fields, the *Companion* decided not to use the *fieldinput* routine.

The selection codes allow you to select records with matching codes. For example, if you coded relatives with the letter R then enter R here, and the program will only print labels for your relatives. If you just press RETURN, entering no codes, the program shown in Figure 10-16 prints mailing labels for all records.

In response to the ARE LABELS IN PRINTER? prompt, you press one of the three function keys. Function III means YES and the program continues to the alignment question. Function key IV means NO and the program waits until you press the YES or the QUIT function. Function key V means QUIT and the program returns to MAILMENU without printing any labels.

In response to the DO YOU NEED ALIGNMENT prompt, you press one of the same three function keys. YES causes the program to print on the daisy-wheel printer five lines of 30 letter X's and a blank

TEXT MODE SCREEN

COLUMNS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
1	M	M	/	D	D	/	Y	Y	M	A	I	L	I	N	G	L	I	S	T	N	A	M	E	←	→	X									
2																																			
3																																			
4																																			
5	E	N	T	E	R	F	I	L	E	N	A	M	E	=	X	←	→	X																	
6																																			
7	E	N	T	E	R	D	R	I	V	E	(1	/	2)	=	X																		
8																																			
9	S	E	L	E	C	T	I	O	N	C	O	D	E	S	=	X	←	→	X																
10																																			
11	A	R	E	L	A	B	E	L	S	I	N	P	R	I	N	T	E	R	?																
12																																			
13	D	O	Y	O	U	N	E	E	D	A	L	I	G	N	M	E	N	T	?																
14																																			
15																																			
16																																			
17																																			
18																																			
19																																			
20																																			
21																																			
22																																			
23																																			
24																																			

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

LINES

FIGURE 10-15 Mailing Label—Screen Layout.

line. Every time you press the YES function you get the alignment pattern again. When you're satisfied with the position of the labels, press the NO function. QUIT terminates the program without printing labels.


```

10 REM   maillabels 01/28/84
1000 GOSUB 8000: REM   get filename and drive
1100 GOSUB 9000: REM   open file
1120 IF NOT found% THEN 1000
1200 GOSUB 8300: REM   questions
1300 FOR record = 1 TO size%
1305 PRINT d$
1310 PRINT d$; "read "; file$; ",R"; record
1320 INPUT " "; record$
1330 GOSUB 3000: REM   selection
1340 IF select% THEN GOSUB 4000: REM   print labels
1360 NEXT record
1600 REM   termination
1610 PRINT d$
1620 PRINT d$; "close "; file$
1630 PRINT labels; " labels printed"
1680 END
3000 REM   selection
3100 select% = false%
3105 IF LEFT$(record$, 1) = "D" THEN RETURN
3110 IF LEN(code$) = 0 THEN select% = true%: RETURN
3120 FOR pos = 160 TO 167
3130 FOR code = 1 TO LEN(code$)
3140 IF MID$(code$, code, 1) = MID$(record$, pos, 1) THEN select% = true%
3150 NEXT code
3160 NEXT pos
3190 RETURN
4000 REM   print label
4020 line = 1: labels = labels+1: PR #1
4030 IF MID$(record$, 2, 29) = " " THEN 4100
4040 name$ = MID$(record$, 2, 12)
4044 FOR long = 11 TO 1 STEP -1
4050 IF RIGHT$(name$, 1) <> " " THEN 4080
4060 name$ = LEFT$(name$, long)
4070 NEXT
4080 line = line+1
4085 IF LEN(name$) <> 0 THEN name$ = name$+" "
4090 PRINT name$; MID$(record$, 14, 17)
4100 FOR pos = 31 TO 91 STEP 30
4120 IF MID$(record$, pos, 30) = " " THEN 4190
4130 PRINT MID$(record$, pos, 30)
4140 line = line+1
4190 NEXT pos
4200 REM   city state zip
4210 city$ = MID$(record$, 121, 14)
4220 FOR long = 13 TO 1 STEP -1
4230 IF RIGHT$(city$, 1) <> " " THEN 4300
4240 city$ = RIGHT$(city$, long)
4300 IF LEN(city$) <> 0 THEN city$ = city$+" "
4310 IF LEN(city$) <> 0 THEN line$ = city$+" "
4320 state$ = MID$(record$, 135, 2)
4330 IF state$ = " " THEN state$ = " ": GOTO 4350
4340 state$ = state$+" "
4350 PRINT city$; state$; MID$(record$, 137, 5)
4360 line = line+1
4400 FOR fill = line TO 6: PRINT: NEXT fill
4410 PR #0
4990 RETURN
7000 REM   errors
7100 HTAB 1: VTAB 22
7210 IF err% = 1 THEN PRINT "TOO SHORT/LONG!";
7220 IF err% = 2 THEN PRINT "ONLY 1 OR 2!";

```

Once you press NO to the alignment question, the program tries to read the file. If the filename does not exist you will get a FILE filename DOES NOT EXIST error message, and you have to start again by entering a new name. You may want to terminate the mailing system and use the CATALOG command to get the proper filename.


```

7240 IF err% = 4 THEN PRINT "INVALID KEY":
7250 IF err% = 5 THEN PRINT "file "; file$; " not found":
7310 IF err% = 9 THEN PRINT "errnum="; ERRNUM(0): STOP
7900 PRINT beep$:
7990 RETURN
8000 REM          display prompt
8010 beep$ = CHR$(7)+CHR$(7): d$ = CHR$(4)
8020 true% = 1
8100 HOME
8105 IF err% = 5 THEN GOSUB 7000
8110 HTAB 9: VTAB 3: PRINT "PRINT LABELS"
8120 HTAB 1: VTAB 5: INPUT "ENTER FILENAME="; file$
8130 IF LEN(file$) = 0 OR LEN(file$) > 10 THEN err% = 1: GOSUB 7000: GOTO 8120

8200 HTAB 1: VTAB 7: INPUT "ENTER DRIVE(1/2)="; drive$
8220 IF drive$ < "1" OR drive$ > "2" THEN err% = 2: GOSUB 7000: GOTO 8200
8290 RETURN
8300 HTAB 1: VTAB 9: INPUT "SELECTION CODES="; codes$
8330 HTAB 12: VTAB 23: PRINT "III IV V";
8400 HTAB 12: VTAB 24: PRINT "YES NO QUIT";
8500 HTAB 1: VTAB 11: PRINT "ARE LABELS IN PRINTER?"
8360 GOSUB 8000
8370 IF NOT ans% THEN 8350
8400 HTAB 1: VTAB 13: PRINT "DO YOU NEED ALIGNMENT?"
8410 GOSUB 8000
8420 IF NOT ans% THEN RETURN
8440 record$ = ""
8450 FOR pos = 1 TO 199: record$ = record$+"x": NEXT pos
8460 GOSUB 4000: REM      print label
8470 HOME: GOSUB 9300: GOTO 8330
8800 REM      get key
8810 GET key$: key% = ASC(key$)
8815 HTAB 1: VTAB 22: PRINT SPC(30);
8820 IF key% = 131 THEN ans% = true%: RETURN
8830 IF key% = 132 THEN ans% = false%: RETURN
8840 IF key% = 133 THEN GOSUB 1600: END
8860 err% = 4: GOSUB 7000: GOTO 8800
8990 RETURN
9000 REM          read control
9190 ONERR GOTO 9700
9200 PRINT d$; "open "; file$; ",L200,D"; drive$
9300 PRINT d$; "READ "; file$; ",R0"
9350 INPUT "": control$
9355 CLRERR: found% = true%
9360 PRINT d$
9370 size% = VAL(MID$(control$, 37, 3))
9380 HTAB 1: VTAB 1: PRINT MID$(control$, 40, 8); " "; MID$(control$, 13, 21);

9990 RETURN
9700 REM      onerr routine
9710 CLRERR: IF ERRNUM(0) = 5 THEN 9800
9720 PRINT "ERRNUM="; ERRNUM(0)
9730 RETURN
9800 REM      file not found
9820 PRINT d$; "close "; file$
9830 PRINT d$; "delete "; file$
9840 err% = 5
9850 RETURN
9990 STOP

```

FIGURE 10-16 MAILLABEL Listing.

If the file exists, the program reads the file sequentially. For records matching the selection codes, it PRINTs labels. You can terminate the label printing at any time by pressing the number sign (#) key on controller 1; the program will close the files, return output to the screen and RUN MAILMENU. Use CONTROL-C to terminate without closing the file or RUNning MAILMENU.

TEXT MODE SCREEN

		COLUMNS																																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
LINES	1	F	I	R	S	T		N	A	M	E				L	A	S	T		N	A	M	E											1	
	2	A	D	D	R	E	S	S		L	I	N	E	-	2																			2	
	3	A	D	D	R	E	S	S		L	I	N	E	-	3																			3	
	4	A	D	D	R	E	S	S		L	I	N	E	-	4																			4	
	5	C	I	T	Y											X		S	T															5	
	6																																		6
	7																																		7
	8																																		8
	9																																		9
	10																																		10
	11																																		11
	12																																		12
	13																																		13
	14																																		14
	15																																		15
	16																																		16
	17																																		17
	18																																		18
	19																																		19
	20																																		20
	21																																		21
	22																																		22
	23																																		23
	24																																		24

FIGURE 10-17 Mailing Label Sample Layout.

Note that PRINT D\$;"PR#1" directs output to the printer. PRINT D\$ directs output back to the screen.

Figure 10-17 shows sample screen layout.

SYSTEM IMPROVEMENTS

Write `MAILLIST` to print a directory of all names on the file. Use `MAILLABEL` as a model. In fact, `SAVE` a copy of `MAILLABEL` with the name `MAILLIST` and just modify the program to print the directory.

As presented here, the *Little Black Book* doesn't reuse deleted records. If you delete a lot of records, the file will eventually get filled with unused holes. To overcome this undesirable feature, add a field to the control record that has the record number of the first deleted record. Link the deleted records together so that the first deleted record has a field with the record number of the second deleted record. Then modify `MAILMAINT` to use a deleted record to add a new person before adding records at the end of the file.

The *Little Black Book* requires you to enter a record number to retrieve information about a person. You could write a program that creates an index file in which each record contains a last name and record number. Then modify `MAILMAINT` to search the index, using the person's last name, to get the record number.

The records in the mailing file have an entry order sequence. The first record added is first, followed by the second record added. To get an alphabetic order by last name you must write a program to sort the records before printing labels or a directory. Donald E. Knuth, in *Sorting and Searching*, published by Addison-Wesley, describes several sorting techniques.

NOW YOU'RE A PROGRAMMER

If you have typed, tested and understood all the programs in this book, you can consider yourself a programmer. You need the seasoning of experience, actually designing and writing your own programs, to become an expert, but you are indeed a computer programmer. Welcome to the club!

ADAM's Future

The computer as we know it has been with us since 1946, when the ENIAC made its first appearance. Just a few years earlier, in 1937, some of the best minds in the world convened to discuss what new technologies the future held in store. Yet when they presented their final report, there was no such thing as a computer on the list.

Today, computers pervade business, industry, science, medicine, government and more. Even though there are many microprocessors—computerlike devices—in our cars and appliances, most Americans don't have an actual computer in their home. Writing in his book *The Computer Revolution*, published in 1962, Edmund C. Berkeley predicted computerized vacuum cleaners and lawnmowers and a computer-controlled heating and cooling system for the home. Although he was insightful and imaginative in many ways, Berkeley regarded the computer as a home laborer, controlling processes as it might in an office or factory. He didn't glimpse the computer's usefulness as a mind extension, a machine capable of enhancing our lives aesthetically and intellectually—and entertaining us as well. In fact, it was another 20 years before there was a practical, affordable computer for the home. ADAM is the first computer you can put to work (or play) in a useful way that doesn't require that you spend thousands of dollars buying it or hundreds of hours learning it.

Video games showed us new ways to stimulate our senses and central nervous system. Already we've seen that a child's sense of logic and reasoning abilities improve by playing maze and adventure games; perhaps the shoot-'em-up games help develop motor skills and strategic thinking, good for riding a bicycle, becoming a skillful negotiator or flying space ships. There is still much more to learn about how video games influence our minds and our life.

We can certainly say the same for the computer itself, which spawned video games in the first place. As we learn more about what computers can do, we learn more about ourselves. And as we explore the neuroscience of the human brain, we see more and more how, unconsciously, the computer was modeled in the brain's image. A computer brain, or a brainlike computer, is still science fiction; in fact, we may not see one in our lifetime—or at all, if certain critics have their way. But we can take advantage of their similarity today, because everything we do with a computer helps us improve, both as people and as a society.

ADAM thrills our senses: it creates powerful sight and sound images, both in games and word processing. As we learn ADAM's keyboard, we grow more dexterous with our fingers. And as we master SmartBASIC and SmartWriter, we learn new mental and intellectual skills that make our life richer and more meaningful. Coleco plans to introduce new software, as well as hardware, that will help ADAM enrich our lives even more. Within a few short years, you may find your family prefers to spend time using ADAM instead of watching television. You may be even more surprised to see your TV, telephone and ADAM connected to each other, doing things you never even imagined!

ADAM SOFTWARE: A BRAVE NEW WORLD

SmartWriter is the heart of ADAM's home information-management system, an ongoing project at Coleco. Once you write any text in SmartWriter, you'll be able to transfer it elsewhere in the system. Write a letter to your relatives telling them about the kids, how old they are and what they're doing in school. You can store names and addresses in the address book so you remember to send Christmas cards. Make a note in your electronic appointment calendar a week before each birthday to remember to buy presents. Save the anecdotes and funny stories you related in each of the kids' electronic diaries you keep. You can do all this by simply touching one or two keys.

Soon SmartWriter will have added features. One is a spelling checker that highlights misspelled words for you. Another is the SmartLETTERS and FORMS package that shows you how to write various kinds of letters, memos and checklists.

ADAM's Family Learning Software will change the way we acquire

knowledge. Programs that re-create storybooks by Dr. Seuss and Richard Scarry help preschoolers and kids in elementary school learn language, literature and geography. Homework Helpers make schoolwork fun for older children. Adults will enjoy learning new things about politics, big business and world affairs with World Shapers and the self-improvement software packages. The whole family will enjoy the Smurf Paint 'n Play Theater, creating their own skits and plays featuring the Smurfs. Brain Strainers help exercise pattern recognition, auditory and visual perceptions, and memorization techniques through playing music games, quizzes and graphics games.

The SmartPICTURE PROCESSOR program gives you powerful graphics for simple drawings, video game graphics, elegant artistic renderings and more, even in three dimensions if you like. You can save anything you draw, such as the Smurf theater skits, video games, a cartoon strip you create showing a space ship exploring outer space, even charts and graphs needed for business, by recording it on your video cassette recorder and playing it back like home movies. As you read further in this chapter, you'll see many more ways your TV and video cassette recorder are natural creative extensions for your ADAM computer.

Making ADAM Smarter

Coleco designed ADAM as a complete, self-contained computer, and it is. That doesn't mean, however, that you can't add other components, called peripherals, that help ADAM do more things.

You've probably read that SmartBASIC is compatible with the Apple computer's BASIC, called Applesoft. Up to this point you've probably said, "So what?" Well, now Coleco has a floppy disk drive for ADAM that changes all that. As the *Companion* explained in Chapter 1, programs may be stored on either digital data packs or mini-floppy disks (the 5¼" size).

There are two advantages in using the disk drive. One, you'll be able to use longer, more complex programs, since the disk holds about twice as much data as the digital data pack. That means you'll be able to store twice as much data of your own, too. Second, the disk is much faster than the digital data pack, so you don't have to wait so long to load programs or find your data. If you really get into using your ADAM, you'll soon find your patience taxed by waiting

several minutes at a time for directory searches and program loading.

In addition, many successful software companies are preparing their programs on floppy disks for ADAM. Infocom, the top adventure game company, joins Spinnaker, Sierra On-Line, Broderbund and others in offering a vast array of great games and programs for your ADAM. Some of these programs are based on the CP/M operating system, the industry standard for personal computers. Needless to say, ADAM has the capability to run CP/M programs with its disk drive and digital data drive, which expands the number of software packages available to you all the more.

Think of it this way: when you bought your ADAM or Expansion Module #3, you got a first-rate game system and ADAM, a computer with word processing and the most "user-friendly" BASIC available. In the future Coleco may offer an electronic card to allow ADAM to run programs for the IBM PC Jr.

The ADAM Family of Languages

SmartBASIC is an excellent language for beginners—in fact, that's why it was called Beginner's All-purpose Symbolic Instruction Code. The *Companion* teaches you how to begin writing programs in BASIC, and Appendix C describes magazines and books that help you learn more. As you grow more proficient, you can graduate to SmartBASIC II, which has advanced graphics, such as sprites, and more sound capabilities.

If you are serious about programming, you'll want to get the add-on 64K Ram Memory Expander, which permits writing longer programs. The extra 64K gives you almost twice as much storage—from 80K to 144K.

But BASIC isn't the only programming language ADAM speaks. Everyone from preschoolers to parents will enjoy programming in SmartLOGO, Coleco's version of the LOGO programming language developed at MIT. Logo is Greek for "word," which is what LOGO employs—plain, everyday English words to write programs. Tots can create programs to direct a turtle to move around the screen. In addition, Coleco plans a ColecoVision game cartridge called Telly Turtle, which teaches LOGO programming using the joystick.

Coleco also plans to introduce Personal CP/M, an advanced, easy-to-use version of CP/M. Knowing how to use the operating system gives you the ability to add special commands or features to pro-

grams, to copy files, and control ADAM and various peripherals you may wish to add (discussed later in this chapter). Also included with Personal CP/M is the CP/M Programmer's Tool Kit that gives you access to the Z-80 microcomputer chip through a *macro-assembler*. This allows you to write new instructions for ADAM's brain in binary, or ones and zeros. This is a job for the more serious and experienced programmer, but it's good to know you'll have this option with ADAM.

Touching the Sky

Some of the most amazing things a computer can do happen once you connect it to the telephone lines. This is called telecommunications, and with the AdamLink 1200 Direct Connect Modem plugged into AdamNet port on your ADAM, you'll discover an endlessly fascinating new world.

A modem is an electronic device that makes it possible to send and receive a computer's electronic blips. It *modulates* them and turns them into a form of electronic code that can be sent over telephone lines and *demodulates* them at the other end. That's where the word *modem* comes from.

Would you like to shop at home? You can with Comp-U-Store, an electronic mail-order store. In the very near future, most of us can pay bills and do our banking at home with ADAM.

If you buy the AdamLink modem, you'll get to use a free sampler at CompuServe, an information utility, including ADAM-ON-LINE. An information utility is like the gas company or the electric company, except that its utility is information. There are two other major information utilities, Delphi (operated by General Videotext in Cambridge, Massachusetts) and The Source (in McLean, Virginia). You reach any of them by dialing a local phone number in your area. Each has varying types of memberships and charges you for the amount of time you use each month. All offer a wide variety of services such as:

- Current business, state, national and international news, including sports and features.
- Stockmarket reports and investor information.
- Travel, hotel and restaurant guides.
- Information databases—electronic libraries you can use to find

specific information for term papers and reports.

- Communication with other members, the computer version of ham or CB radio conversations.

- Games you can play and games you can have "downloaded," or sent to your ADAM and recorded on tape or disk.

In addition, if you're retrieving information such as an article from a magazine or something from an encyclopedia from an information database, you can download it to your printer and thus read it at your convenience. Information utilities are very useful, and they provide many more services, such as self-publishing, storing information, an appointment calendar, newsletters, even banking and shopping, that you'll want to explore yourself.

Another special information utility is the game network Bell Labs is working toward creating in conjunction with Coleco. Some of the games are Coleco's and others are designed by the engineers at Bell Labs, but all of them may be played over the telephone lines with your ADAM. Some are *interactive* games, which means you are one of several—perhaps even hundreds—of other players all across the country, all playing the game together!

As you become more proficient with your ADAM and modem, you'll be able to call friends who have an ADAM and modem and communicate directly with them. You can trade programs, games and information among your computers.

Mind Extensions

The AdamNet (next page) port on ADAM's left side is not unlike the Biblical Adam's rib: it is a gateway to unlocking ADAM's potential even more. It is possible to hook up other connector boxes that help ADAM do more things.

- One modem may be permanently connected to your information utility, while another is open to communicating with your friends who have ADAMs. Or you may want to set up an ADAM computer bulletin board for your local ADAM User Group, with information about meetings, new games and special announcements from other users.

- While Coleco has no plans to market an RS-232 interface, the *Companion* expects other vendors to market one. Via the RS-232 serial

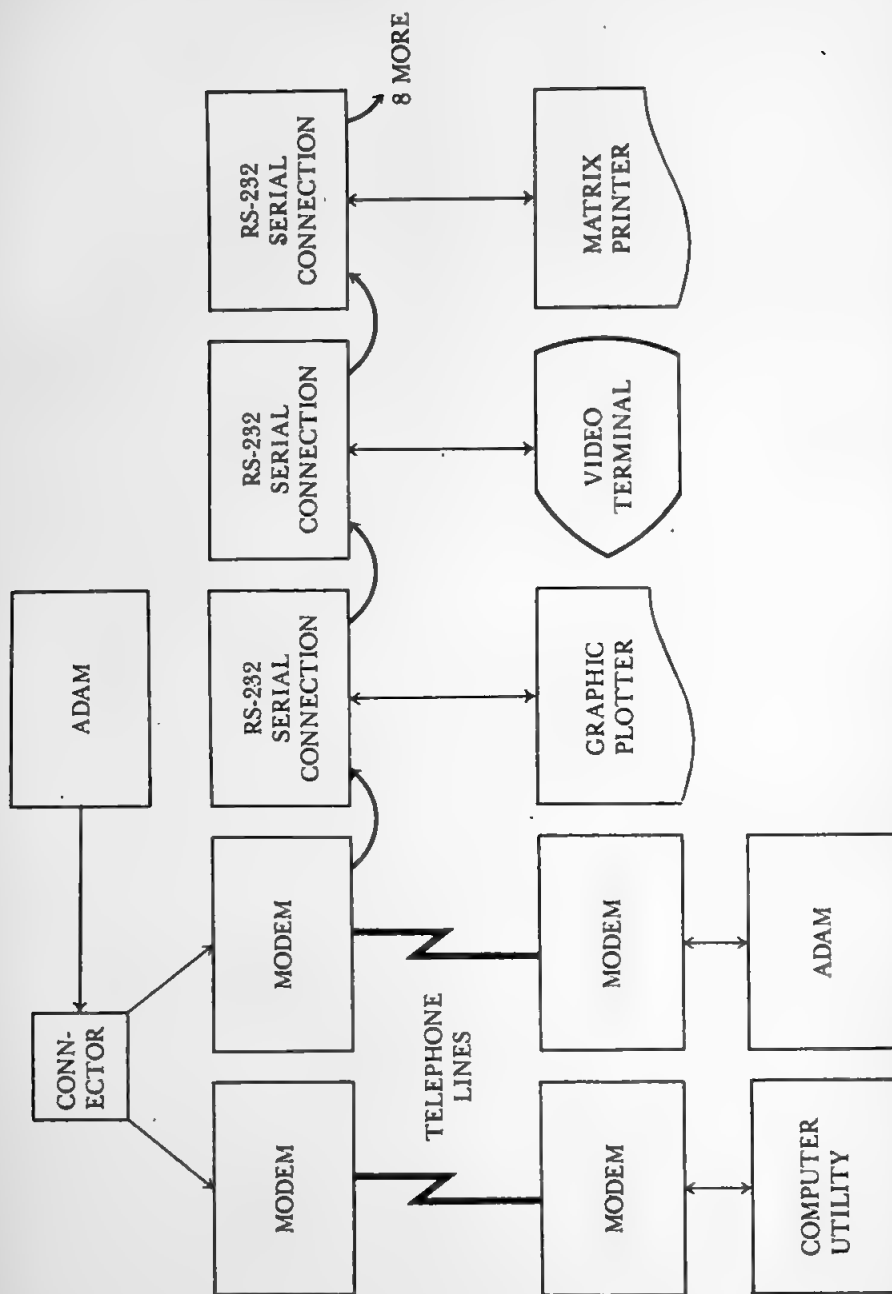


FIGURE 11-1 AdamNet.

connector box, you may connect other printers to your ADAM. You may want a faster *dot-matrix* printer, or perhaps a *color graphics plotter*. If more than one person in the family uses ADAM, you can connect *remote terminals*, another keyboard or monitor, so dad, big sister and little brother can all use ADAM without unplugging it and lugging it from room to room. Schools will find this a valuable feature, too.

With the proper connections, you'll be able to connect ADAM through your video system and use it in new ways. ADAM can be a controller for making video movies combining computer graphics, videodisk scenes, live television and audio together on videotape. You may also link ADAM to the television as a videotext terminal, if such services are available in your area.

ADAM would be happy to be a sentinel for your home. With the addition of the real-time clock, and appropriate interface circuitry, ADAM can turn lights on and off at dawn and dusk, start the coffee in the morning, and watch the house when you're away. If a burglar broke in and tripped the circuit breaker ADAM would detect it, and would call the police department and play a prerecorded tape message over the phone. Similarly, ADAM could detect a fire and call the fire department.

In time, you'll be able to give ADAM instructions verbally rather than having to use the keyboard. ADAM may even talk back to you, answering your questions or informing you when a task is finished. Many computers were designed to do just one or two things, such as play games; Coleco designed ADAM as a full-function computer with infinite expandability.

1984 will go down in the history of computing as the year of ADAM, the family home computer. If you are an ADAM owner, you'll be part of this history. You are one of the elite new members of the "information society," people who want to learn about computers instead of fearing them. Hank Koehn, vice-president of Futures Research at Security Pacific Bank in Los Angeles, says,

We live in a society where people prefer visual information. The electronic media—mating information delivery systems with video technology and computers—have made people think about things they never would have thought about before. If the computer can turn people on to new ideas and new possibilities, then it's really enhancing them.

Welcome to the brave new world of computers—and ADAM!

APPENDIX A

Description of ASCII Character Codes

<i>ASCII Value</i>	<i>Display Character</i>	<i>Keystroke</i>
0	[See page 274 for symbols illustrating 1-31]	Control-@
1		Control-A
2		Control-B
3		Control-C
4		Control-D
5		Control-E
6		Control-F
7		Control-G
8		Control-H
9		Control-I
10		Control-J
11		Control-K
12		Control-L
13		Control-M
14		Control-N
15		Control-O
16		Control-P
17		Control-Q
18		Control-R
19		Control-S
20		Control-T
21		Control-U
22		Control-V
23		Control-W
24		Control-X
25		Control-Y
26		Control-Z
27		n.a
28		n.a
29		n.a
30		n.a
31		n.a

32	Space	Space Bar
33	!	!
34	"	"
35	#	#
36	\$	\$
37	%	%
38	&	&
39	'	'
40	((
41))
42	*	*
43	+	+
44	,	,
45	-	-
46	.	.
47	/	/
48	0	0
49	1	1
50	2	2
51	3	3
52	4	4
53	5	5
54	6	6
55	7	7
56	8	8
57	9	9
58	:	:
59	;	;
60	<	<
61	=	=
62	>	>
63	?	?
64	@	@
65	A	Shift-A
66	B	Shift-B
67	C	Shift-C
68	D	Shift-D
69	E	Shift-E
70	F	Shift-F
71	G	Shift-G
72	H	Shift-H
73	I	Shift-I
74	J	Shift-J
75	K	Shift-K
76	L	Shift-L

77	M	Shift-M
78	N	Shift-N
79	O	Shift-O
80	P	Shift-P
81	Q	Shift-Q
82	R	Shift-R
83	S	Shift-S
84	T	Shift-T
85	U	Shift-U
86	V	Shift-V
87	W	Shift-W
88	X	Shift-X
89	Y	Shift-Y
90	Z	Shift-Z
91	[[
92	\	\
93]]
94	^	^
95		
96		
97	a	A
98	b	B
99	c	C
100	d	D
101	e	E
102	f	F
103	g	G
104	h	H
105	i	I
106	j	J
107	k	K
108	l	L
109	m	M
110	n	N
111	o	O
112	p	P
113	q	Q
114	r	R
115	s	S
116	t	T
117	u	U
118	v	V
119	w	W
120	x	X
121	y	Y
122	z	Z

123	
124	
125	
126	
127	
128	Home
129	Function I
130	Function II
131	Function III
132	Function IV
133	Function V
134	Function VI
135	
136	
137	Shift and Function I
138	Shift and Function II
139	Shift and Function III
140	Shift and Function IV
141	Shift and Function V
142	Shift and Function VI
143	
144	Wild Card
145	Undo
146	Move/Copy
147	Store/Get
148	Insert
149	Print
150	Clear
151	Delete
152	Shift and Wild Card
153	Shift and Undo
154	Shift and Move/Copy
155	Shift and Store/Get
156	Shift and Insert
157	Shift and Print (Screen Print?)
158	Shift and Clear
159	
160	Up arrow
161	Right arrow
162	Down arrow
163	Left arrow
164	Control and up arrow
165	Control and right arrow
166	Control and down arrow
167	Control and left arrow

0		16	
1	-	17	≧
2	☹	18	≦
3	🏠	19	±
4	♥	20	🎵
5	♦	21	🎵
6	🔑	22	
7		23	✦
8		24	
9		25	🎵
10		26	🎵
11	Σ	27	➡
12		28	
13		29	⌋
14	✓	30	ℱ
15	∞	31	⌞

FIGURE A-1 ASCII Display Characters 1-31.

SmartBASIC Command and Function Summary

ABS (arithmetic expression)

The ABSolute function allows you to make sure that the value of the arithmetic expression is positive by removing the minus sign from a negative value. If the value of the expression is already positive, the function has no effect.

Immediate-mode example:

You type PRINT ABS (+5)

ADAM displays 5

If the value of the expression has a negative sign, however, the ABS function changes the sign to positive.

You type PRINT ABS(5-10)

ADAM displays 5

If you didn't use the ABS function, see what would happen.

You type PRINT 5-10

ADAM displays -5

Program example:

Use the ABS function to toggle a switch on and off.

```
10 REM  abs toggle switch on(1) and off(0)
1000 FOR count = 1 TO 5
1100 sw% = ABS(sw%-1)
1200 PRINT sw%
1300 NEXT count
```

Helpful hints:

Use the SGN function if you want to check the sign before using the ABS function.

AND

You use AND, called a logical operator, to test if two values are both true. A true result returns a value one, and a false result returns a value zero. The test, called a compound logical expression, compares either numeric variables (for example, *switch%*), arithmetic expression (for example, *switch% - 3*) or relational tests (for example, *switch% > 5*). The following truth table summarizes the possible combinations and results.

<i>Value 1</i>	<i>AND</i>	<i>Value 2</i>	<i>Result</i>
true		true	true
false		true	false
true		false	false
false		false	false

Immediate-mode example:

Try the following examples of compound logical expressions formed with the AND operator.

You type ?1 AND 1

ADAM displays 1

The value one means true. True and true gives a true result.

You type true%=1

You type false%=0

You type PRINT true% AND false%

ADAM displays 0

The value zero means false. True and false gives a false result.

You type ?true% AND 5=4

ADAM displays 0

A false result because the logical relation $5 = 4$ is false. True and false gives a false result.

You type ?4 AND 5

ADAM displays 1

A true result because any nonzero value is true.

Program example:

The following program loops until you press both buttons on controller 1. Reading a PDL function value resets its value to zero.

```

10 REM and
1000 REM loop
1100 left% = PDL(7)
1200 right% = PDL(9)
1250 PRINT "LEFT="; left%; " RIGHT="; right%
1500 IF left% AND right% THEN END
1600 GOTO 1000

```


Helpful hints:

Normally you use the AND logical operator in an IF . . . THEN statement to determine the combined truth of two values.

Refer to OR, and NOT, the other logical operators.

Unless otherwise indicated by parentheses, BASIC performs AND comparisons before OR comparisons.

For example,

You type ?1 AND 1 OR 0 AND 0

ADAM displays 1

The ANDs done first give one or zero, which is true. But if you use parentheses to do the OR first,

You type ?1 AND (1 OR 0) AND 0

ADAM displays 0

A zero results because one and one and zero is false.

APPEND filename,Ddrive

You use the APPEND command to add data to the *end* of an existing sequential file. The command takes the place of the OPEN command. BASIC keeps track of where you are in the file by using a pointer. While the OPEN command positions the pointer to the beginning, the APPEND command positions the pointer at the end of the file.

Ddrive. The D identifies the parameter as a drive number. The left tape drive is 1, and the right tape drive is 2. The disk drive is 3. If the number is omitted, BASIC defaults to the drive used last.

Program example:

The first short program creates a file named *companion* with the OPEN command. The second program APPENDs a single record to the file.

```
10 REM      append
100 d$ = CHR$(4)
1000 PRINT d$; "append program"
1200 PRINT "20 rem line 20"
1300 PRINT d$; "close program"
```

Helpful hints:

Always remember to CLOSE a file.

On the tape cassette, BASIC extends the file by making a copy of the existing file and adding the new data to the end of the copied file. For a file containing several records, this can take more than just a few minutes.

Use the OPEN command to create a new file.

ASC (string expression)

You use the ASC function to get the decimal ASCII value for the first character in the string argument.

Immediate-mode example:

You type PRINT ASC("ABC")

ADAM displays 65

The decimal number 65 represents the letter A.

Program example:

The following program reads a character from the keyboard and displays its ASCII value. With the GET statement, BASIC doesn't wait for you to hit the RETURN key. It passes the typed character right to the BASIC program, and BASIC stores it in *key\$*, a string variable. To terminate the program, press the CONTROL key and the letter C key at the same time (CTL-C), which sends the value 3 to the computer.

```
10 REM asc
1000 GET key$
1100 ascii% = ASC(key$)
1200 PRINT "CHR="; key$; " ASCII VALUE="; ascii%
1400 IF ascii% <> 3 THEN 1000
```

Notice what happens when you press the function key. With many keys, if you press the shift you will generate a different value. Appendix A lists the value generated by each key.

Helpful hints:

Remember, this function returns the value of only the first character in a string. A null string—a string having no characters in it—terminates the program and BASIC displays the following error message:

?ILLEGAL QUANTITY

To convert an entire string to a decimal value, use the VAL function.

To convert a number to a character, use the CHR\$ function.

ATN (arithmetic expression)

Returns the arctangent, in radians, of the value of the argument in the parentheses. The angle returned approaches $+\pi/2$ radians as the argument approaches $+\infty$ and $-\pi/2$ radians as the argument approaches $-\infty$.

Immediate-mode example:

You type ?ATN(1)

ADAM displays .785398164 (in radians)

Program example:

Pi (π), the ratio of the circumference of a circle to its diameter, has an approximate constant value for any circle of $\pi = 3.14159265$. Radians convert to degrees by the formula $2 * \pi$ radians = 360 degrees, which means $\pi/180 = 1$ degree. The following prints the radians and degrees for three argument values, -1, 0, +1.

```

10 REM  atn
500 pi = 3.14159265
510 factor = 180/pi
1000 PRINT "ARC TANGENT CALCULATOR"
1100 INPUT "RADIANS="; arc
1200 GOSUB 2000
1300 GOTO 1100
2000 REM  print arc tangent
2100 degrees = ATN(arc)*factor
2120 PRINT "ARC="; arc; " ATN="; ATN(arc); " DEGREES="; degrees
2190 RETURN

```

Helpful hints:

Refer to the other trigonometric functions SIN, COS and TAN.

BLOAD filename,Aaddress,Ddrive

This command loads, from tape or disk, a binary-memory image file into memory. This file is usually a shape table or user-written Z-80 microprocessor-language program. The shape table must be created using the BSAVE command.

Example:

```
BLOAD shapetable,A28000,D1
```

The *filename*, here *shapetable*, is required and must be a standard ADAM filename (1 to 10 characters long) with no embedded spaces.

Address. The A identifies the parameter as an address. If A is omitted, SmartBASIC assumes a default, to the address you used when you did the BSAVE to create it. In the example, you tell BASIC to load the shape table into decimal address 28000.

Drive. The D identifies the parameter as a drive number. The left tape drive is 1, and the right tape drive is 2. If the number is omitted, it defaults to the drive used last.

```

10 REM  bload
20 LOMEM :29000
100 DIM grid%(15)
1350 d$ = CHR$(4)
1400 PRINT d$; "bload row,A28000,D1"

```



```

2000 location = 28000
2050 FOR row = 0 TO 15 STEP 2
2100 byte% = grid%(row)+16*grid%(row+1)
2200 byte% = PEEK(location)
2210 second% = byte%/16
2220 grid%(row+1) = second%
2230 grid%(row) = byte%-second%*16
2250 location = location+1
2300 NEXT row
3000 GR
3100 column = 20
3500 FOR row = 0 TO 15
3600 COLOR = grid%(row)
3800 PLOT column, row
3900 NEXT row

```

Helpful hints:

BLOADing a file takes less time than reading a sequential file. You can modify PICMAKER to POKE the *grid%* array into memory locations. You may want to save several pictures in contiguous memory locations, copy them all to tape with BSAVE, and then load them back into memory with BLOAD. Your slide show will flip from picture to picture much faster, because you are retrieving the pictures from memory instead of from slow tape.

BRUN filename,Aaddress,Ddrive

BRUN loads, from tape or disk, a binary-memory image file and then automatically executes it.

Helpful hints:

Although this command is available, it is useless without a Z-80 assembler to create the Z-80 programs. Hand coding a microprocessor program is virtually impossible. Wait for Coleco to release its Z-80 macro-assembler.

BSAVE filename,Aaddress,Llength,Ddrive

This command saves a binary-memory image file on tape or disk. You use the BSAVE to create a file that BLOAD reads back into memory.

Example:

BSAVE shapetable,A28000,L100,D2

The *filename*, here *shapetable*, is required and must be a standard ADAM filename (1 to 10 characters long).

Address. The A identifies the parameter as an address. It defines the starting memory address of the binary data you want saved. In the example, you tell BASIC to save the beginning at decimal memory address 28000.

Llength. The L identifies the parameter as a length, the number of bytes you want to save. In the example, you are saving 100 bytes or characters beginning at memory address 28000—in other words, the bytes from 28000 to 28099.

Ddrive. The D identifies the parameter as a drive number. The left tape drive is 1, and the right tape drive is 2. If the number is omitted, BASIC defaults to the drive used last.

```

10 REM  bsave
20 LOMEM :29000
30 address = 28000
100 DIM grid%(15)
1000 GR
1100 column = 20
1500 FOR row = 0 TO 15
1600 grid%(row) = row
1700 COLOR = row
1800 PLOT column, row
1900 NEXT row
2000 location = address
2050 FOR row = 0 TO 15 STEP 2
2100 byte% = grid%(row)+16*grid%(row+1)
2200 POKE location, byte%
2250 location = location+1
2300 NEXT row
2350 d$ = CHR$(4)
2400 PRINT d$; "bsave row,A"; address; ",L8"

```

Helpful hints:

Refer to BLOAD.

Use LOMEM: to reserve the memory space and the POKE command to move the data. Refer to DRAW for an example of saving a shape table.

CALL (arithmetic expression)

The CALL statement allows you to call or execute a Z-80 microprocessor subroutine that has been stored beginning at the memory location defined by the value of the arithmetic expression. While the memory location can range from 0 to 65535, SmartBASIC resides in the first 27999 decimal locations. At this time, Coleco has not released the addresses of any routines in SmartBASIC that you can call directly. You can store your own Z-80 programs beginning at 28000 by setting LOMEM: above the end of the program. SmartBASIC saves all Z-80 registers before executing your program so you can use all 16. You must return to BASIC with a Z-80 RETURN operation code, decimal value 201.

Program example:

The explosive sound of a gunshot increases the playing excitement of any shoot-'em game. The following program uses the sound routine explained in Chapter 9 to program that noise channel to create the sound of an explosion. The decimal value 228 programs the noise channel for continuous white (random vibrations) noise. The decimal value 224 would select a periodic noise. The FOR . . . NEXT loop is initialized with 240, the loudest noise sound value, and each higher step value reduces the volume until 255 shuts it off.

```

5 LOMEM :29000
10 REM call explosion
1100 GOSUB 16700: REM init sound values
1200 POKE chip%, 228: CALL sound%
2200 FOR loud = 240 TO 255
2220 POKE chip%, loud: CALL sound%
2260 FOR delay = 1 TO 40: NEXT delay
2280 NEXT loud
2350 GET key$
2360 IF ASC(key$) = 3 THEN END
2990 GOTO 1200
16700 REM sound
16705 DATA 58,102,109,211,255,201
16730 sound% = 28000
16740 chip% = 28006
16810 FOR address = sound% TO sound%+5
16820 READ byte%
16830 POKE address, byte%
16840 NEXT address
16990 RETURN

```

Helpful hints:

Refer to the description and tables in Chapter 8 and experiment with the different sounds you can make with ADAM and SmartBASIC.

BASIC, although powerful and easy to use, has limitations. You can do things with Z-80 microprocessor instructions that you cannot do in BASIC because of its slow execution speed relative to machine-language instructions. To learn about programming the Z-80, *Adam's Companion* recommends *Programming the Z-80* by Rodney Zaks, published by Sybex. Look for information in *ADAM Family Computing* for how to call useful routines in SmartBASIC and OS-7, ADAM's operating system.

CATALOG Ddrive

You use this command whenever you want to know what files are on a tape or disk. It displays a directory showing for each file:

Type: Identifies the version and language that created the file. A capital

letter like H or A identifies the current version, while a lowercase letter identifies the backup version. The table below summarizes the type codes used by SmartWriter and SmartBASIC. (The LOGO language files, for example, will use an as yet unspecified code.

H	SmartWriter
h	SmartWriter backup file
A	SmartBASIC program or data file
a	SmartBASIC backup file

Sectors. Number of 1024-character blocks used by file.

Filename. Name of file from 1 to 10 characters.

Immediate-mode example:

You type	CATALOG D1
ADAM displays	VOLUME: FIRST DIR
	A 1 hello
	H 2 letter

This example shows that the tape contains two files: *hello*, a SmartBASIC program, and *letter*, a file created by SmartWriter

To print catalog:

You type	pr#1 (turns on printer)
You type	catalog
You type	pr#0 (turns on printer)

ADAM prints the catalog on the daisy-wheel printer.

Helpful hints:

Always use CATALOG before initializing tape with the INIT command, in case you put the wrong tape in the drive.

CHRS (arithmetic expression)

The CHRS statement allows you to convert a number to a character. Since a byte stores characters and an 8-bit byte can only store 0 to 255 values, the value of the argument inside the parentheses must be in the range 0 to 255.

Immediate-mode example:

You type	PRINT CHRS(66)
ADAM displays	B

Program example:

The following program prints the 96 displayable characters on the daisy-

wheel printer. It prints 96 lines, so make sure you have continuous forms in the printer. (A single sheet has room for only 66 lines.)

```

10 REM chr print ascii characters and values
100 ONERR GOTO 3000: REM ctl-c turns off printer
500 PR #1: REM turn on printer
1000 PRINT "NUMBER CHR$"
2000 FOR ch = 32 TO 127
2100 PRINT ch; TAB(8); CHR$(ch)
2200 NEXT ch
3000 PR #0: REM turn off printer

```

Helpful hints:

Appendix A lists all the ASCII characters and their numeric values.

The ASC function converts a character back to a number.

Use the CHR\$ to define nondisplayable characters such as the file-output-control character, as done in statement 100 in the example above.

CLEAR

You use CLEAR to reset every variable in your program to zero and make every string a null string. Normally you don't use this statement because BASIC resets all variables before running your program. Sometimes you may want to repeat your program, however. Rather than reRUN the program, you can execute a CLEAR statement at the beginning of your program so that you know that every variable is correctly initialized.

Immediate-mode example:

```

You type      A=10
You type      PRINT A
ADAM displays 10
You type      CLEAR
You type      PRINT A
ADAM displays 0

```

The CLEAR statement resets the variable A to zero.

Program example:

The following program uses the CLEAR command to set the variable *amount\$* to a null string and the variable *total* to zero.

```

10 REM clear
1000 CLEAR
1500 ONERR GOTO 6000
2000 INPUT "AMOUNT="; amount$
3000 total = total+VAL(amount$)
4000 PRINT "total="; total
5000 GOTO 2000
6000 CLRERR
6100 INPUT "Restart(Y/N)?"; ans$
6200 IF ans$ = "y" OR ans$ = "Y" THEN 1000

```


Helpful hints:

Always place the CLEAR statement at the beginning of the program so the reader of the program knows it is there.

CLOSE filename

With this file command you tell BASIC you no longer need to read data from or write data to a tape or disk file. You give the command to BASIC by using the PRINT command. The text in the PRINT command consists of CONTROL-D (a nondisplayable ASCII character with a decimal value of 4) and the word CLOSE (in capital letters). By convention the string variable DS is used to store the value for CONTROL-D. The program-initialization section assigns DS=CHRS(4).

Immediate-mode example:

When you OPEN a file, BASIC doesn't know how large a file you will write, so it allocates all available space on the tape. If the program terminates before closing the file, you may have to close the file, and BASIC shortens the file to the actual room used; otherwise the next file you try to create will find no more room on the tape.

Program example:

This program illustrates how to create a sequential file.

NEW

10 REM CLOSE

1000 DS=CHRS(4)

2000 PRINT DS;"OPEN ADAM-HELP"

2100 PRINT DS;"WRITE ADAM-HELP"

2200 PRINT "ASC-value of a character"

2300 PRINT "ABS-make negative number positive"

3000 PRINT DS;"CLOSE ADAM-HELP"

RUN

See the description of the OPEN command for an example that will create the file and print it.

Helpful hints:

Closing a file frees the memory buffer space so you can open another file.

Refer to OPEN, APPEND, READ and WRITE.

CLRERR

When you no longer want to handle errors with the program's ONERR routine, use this command.

Helpful hints:

Refer to the ERRNUM function and the ONERR command for examples of how to use CLRERR.

COLOR= (arithmetic expression)

You use the **COLOR** statement to set the color plotted by the BASIC low-resolution graphics statements **PLOT**, **VLIN** and **HLIN**. The table below lists the names and numbers of the 16 colors.

<i>Number</i>	<i>Color</i>	<i>Number</i>	<i>Color</i>
0	Black	8	Light yellow
1	Magenta	9	Medium red
2	Dark blue	10	Gray-2
3	Dark red	11	Light red
4	Dark green	12	Light green
5	Gray-1	13	Light yellow
6	Medium green	14	Cyan (blue)
7	Light blue	15	White

Immediate-mode example:

The following statements draw a solid light-red bar on the top line of your color television or monitor.

```
GR
COLOR=11
HLIN 0,35 AT 0
```

Program example:

The following program flashes a large letter **T** on the video screen fifteen times, once for each of the low-resolution colors except black. By switching back and forth between the **GR**aphics display mode and the **TEXT** display mode the program creates the flashing effect. The **FOR . . . NEXT** delay loop keeps the letter on the screen long enough for you to see it.

```
10 REM color
1100 FOR hue = 1 TO 15
1200 GR
1250 COLOR = hue
1300 HLIN 15, 24 AT 6
1400 VLIN 7, 27 AT 20
1450 FOR delay = 1 TO 300: NEXT delay
1500 TEXT
1600 NEXT hue
```

Helpful hints:

The **GR** statement sets **COLOR** to zero (black). Always set the **COLOR** after a **GR** command.

Use the SCRN statement to find the color of a given point on the low-resolution screen. See SCRN for an example.

If you try to set the color to a number with a decimal fraction, BASIC will convert the number to an integer. For example, 4.99 sets the COLOR=4 dark green, not gray (5).

See Chapter 7 for a full explanation of low-resolution graphics programming.

CONT

Use this command to restart a program you have halted by a STOP, END or pressing both the CONTROL key and the letter C key at the same time (CTL-C).

Immediate-mode example:

```
You type      NEW
You type      10 REM CONTINUOUS LOOP
You type      1000 A=A +1
You type      1100 GOTO 1000
You type      RUN
You press     CTL-C
ADAM stops at line 1000
You type      PRINT A
```

By pressing CTL-C you stopped the program's execution. Note the value displayed. Now CONTinue the execution of the program.

```
You type      CONT
You press     CTL-C
You type      ?A
```

The higher value indicates that the program continued execution after you typed the CONT command.

Helpful hints:

When trying to debug a program, you should use STOP statements within the program to halt execution at certain critical points so you can look at the values of the program variables. You use the CONTinue statement to resume execution of the BASIC program. See STOP for an example.

CONT resumes execution of the program at the next command, not necessarily the next line. For example, in the following program line, BASIC executes the PRINT command, then the INPUT statement after you type CONT.

```
3000 A=5
4000 STOP:PRINT A
5000 INPUT "A=";A
```


COS (arithmetic expression in radians)

Trigonometry defines the cosine of an angle as the following ratio:

$$\text{COS } \theta = \frac{\text{adjacent side}}{\text{hypotenuse}}$$

For an angle defined in radians, the COSine function returns the value of this ratio.

Immediate-mode example:

Suppose you want a scarf to fold into a triangle to use as a bandana when you jog. You figure you need 12 inches to go around your head and for the knot. How large a square scarf do you need? The hypotenuse, the longest side of the triangle, is the fold of the scarf, and it must be 12 inches. Each corner of the unfolded scarf is a right angle or 90 degrees. Folding the scarf cuts the angle in half to 45 degrees. Radians = degrees multiplied by pi/180. Now let ADAM calculate the side of the scarf.

You type pi=3.14159265

You type PRINT 12*(COS(45*pi/180))

ADAM displays 8.48528156

A 9-inch-square scarf will work fine.

Program example:

This program plots the curve of the COS curve from 0 to 360 degrees on the high-resolution screen. Since the COS value ranges between 1 and -1, the equation in line 2100 adds 1 so the range becomes 2 to 0. To convert that range into the range of rows 0 to 191, multiply by 96. High-resolution graphics defines the top row as 0 and the bottom as 191. You are used to seeing the Y-axis have 0 at the bottom. So the curve doesn't appear upside down, the value of the point is subtracted from 191. For example, a point at 0 would normally plot at the top row; by subtracting it from 191, it plots at the bottom row.

```

10 REM cos
1000 pi = 3.14159265
1100 HGR: HCOLOR = 9
1200 HPlot 0, 80 TO 255, 80
2000 FOR degree = 0 TO 360 STEP 2
2100 cosine = COS(degree*pi/180)+1
2300 row = 159-(80*cosine)
2400 HPlot degree/2, row
2900 NEXT degree
3000 HTAB 1: PRINT "0": HTAB 20: PRINT "360"
3100 PRINT "PLOT OF COSINE - IN DEGREES"
3200 GET key$
3300 TEXT

```


Helpful hints:

See the SIN function for more details.

If you want to know more about what you can do with trigonometric functions, see "Helpful hints" under TAN.

DATA (list of constants separated by commas)

Use the DATA statement when you want to define data (character strings, decimal numbers or integer numbers) as part of the program. You separate each item in the list by using a comma (,).

When the program loads into memory, BASIC creates a list of constants using the DATA statements in statement-number order. The READ statement moves the constants into the variables named in the READ statement. Because BASIC creates the data list as the program is loaded, not when it is executed, the DATA statement has to come before the READ statement. It can appear anywhere in the program. After the READ statement reads the list once, use the RESTORE statement to tell BASIC to start at the beginning of the list to satisfy the next READ statement's request for data.

Program example:

The example below shows the four types of data you can define with a DATA statement. You can define literals, strings, integers and decimal-point numbers.

```

10 REM data
900 TEXT
1000 FOR hue = 0 TO 3: READ hue$(hue): NEXT hue
1100 DATA BLACK, GREEN
1200 DATA "VIOLET", "WHITE"
1300 DATA 0, 1, 2, 3
1500 READ a%, b%, c%, d%
1550 PRINT a%, b%, c%, d%
1600 READ w, x, y, z
1700 DATA 1.34, 4.56, 543, -98.34
1800 PRINT "W="; w
1810 PRINT "X="; x
1820 PRINT "Z="; z
2000 FOR hue = 0 TO 3: PRINT hue$(hue): NEXT hue
2100 INPUT "Do Again(Y/N)?"; ans$
2200 IF ans$ = "Y" OR ans$ = "y" THEN RESTORE: GOTO 1000

```

Respond Y if you want to make sure RESTORE works.

Helpful hints:

Never use a colon (:) to continue another statement with a DATA statement line. The following statement is *wrong*.

```
100 DATA, RED, GREEN,BLUE : REM COLORS
```

When defining a colon or comma as data, put it inside the quotes of a string.

```
100 DATA " , " , " : "
200 READ comma$,clon$
300 PRINT comma$,clon$
```

You can omit data and BASIC will assign the variable a zero or null string value. See READ and RESTORE for more explanation and examples.

Do not use double quotes (") as data in a DATA statement. The following is *wrong*:

```
DATA " " "
```

DEF FN (function definition)

This command allows you to define your own arithmetic function. You want to do this when you have a formula like *amount * 5*—multiply a number by 5—that you use at different places in your program. Instead of writing it each time, use the DEFinition FuNction command to give the formula a name, specify the argument used in the formula and define the formula. For example, name the formula *timesfive*, which uses the argument *amount* to calculate the formula *amount * 5*.

```
1000 DEF FN timesfive(amount) = amount*5
```

Then anywhere in your program you want to multiply a number by five, use the letters FN, the function name and in parentheses the argument—the value you want multiplied by five.

For example, to print 100 multiplied by 5 you write:

```
2000 PRINT FN timesfive(100)
```

BASIC must execute the DEF FN command before you can reference the defined function.

Immediate-mode example:

You can't use DEF FN in the immediate mode.

Program example:

Suppose you need to calculate interest at several places in your program using different values for principal. The following program defines the

formula for interest and the formula to calculate the interest at 8 percent for \$1000 and \$2000.

```
10 REM def
50 DEF FN interest(principal) = principal*rate
60 rate = .08
1000 PRINT "PRINCIPAL="; 1000
1100 PRINT "INTEREST="; FN interest(1000)
2000 PRINT "PRINCIPAL="; 2000
2100 PRINT "INTEREST="; FN interest(2000)
```

Helpful hints:

You will rarely use the DEF FN command, since a formula must be defined as a single arithmetic expression. The GOSUB and RETURN commands provide a way of executing a subroutine, from different places in your program, that can have one or many statements to define the desired calculations

DEL from-line,to-line

When you want to delete a range of lines from your BASIC program you use this statement.

Immediate-mode example:

You type 100 REM line 1

You type 200 REM line 2

You type 300 REM line 3

You type LIST

ADAM displays 100 REM line 1

200 REM line 2

300 REM line 3

You type DEL 100,200

You type LIST

ADAM displays 300 REM line 3

You type DEL 300

You type LIST

ADAM displays nothing (you deleted all the lines)

Although you can delete statements from within a program, the *Companion* strongly frowns on this technique. After deleting the statements, program execution stops.

Helpful hints:

You can delete a single statement by typing the line number all by itself.
 You type 100 REM line 1
 You type LIST
 ADAM displays 100 REM line 1
 You type 100
 You type LIST
 Adam displays nothing (you deleted the line)

DELETE filename

You use the DELETE command to delete a file from the tape or disk.

Immediate-mode example:

To delete the filename *goodbye*,
 You type DELETE goodbye

Program example:

If you try to open a file that doesn't exist because you spelled the name wrong, BASIC will create an entry in the directory for the misspelled filename and allocate a storage block of the tape or disk for the empty file. The following program shows you how to use the ONERR GOTO command to execute the file-deletion routine when you read an empty file.

```

10 REM delete
100 DIM statement$(100)
500 d$ = CHR$(4)
1000 INPUT "FILE NAME="; filename$
1100 IF LEN(filename$) = 0 THEN END
1200 ONERR GOTO 2000
1300 PRINT d$; "open "; filename$
1400 PRINT d$; "read "; filename$
1600 INPUT " "; statement$(subscript)
1700 subscript = subscript+1
1900 GOTO 1600
2000 CLRERR
2100 IF subscript = 0 AND ERRNUM(0) = 5 THEN 3000
2200 PRINT d$; "close "; filename$
2300 FOR line = 0 TO subscript-1
2400 PRINT statement$(line)
2500 NEXT line
2900 END
3000 PRINT d$; "close "; filename$
3100 PRINT d$; "delete "; filename$
3200 PRINT "FILE "; filename$; " NOT FOUND"
3900 GOTO 1000

```


Helpful hints:

See the RECOVER command to see how you can make the backup file the current file.

Before deleting a BASIC program file, LOAD and LIST the program to make sure you really want to delete it.

DIM variable (size of array)

You use the DIMension statement to define the size and dimension of an array. While BASIC stores one value at a memory location named by the variable, you use arrays to store a collection of values all with the same name. For example, while DAYS stores the name of one day, like Sunday, to store the collection of names of all seven days of the week, you would DIMension the array DAYS(7), which actually provides space for eight items or array elements—from zero to seven. You use a numeric value called a subscript to reference each element in the array. A one-dimensional array has only one subscript, a two-dimensional array has two subscripts, etc. The example shows you can have string, real-number and integer arrays.

```
100 DIM DAYS(7),ROWCOL(10,10),LWH%(3,4,2)
```

Array subscripts range from zero to the size you specify, so even if you specify zero for the subscript size BASIC sets up one element in storage.

You will find that arrays provide a powerful storage method allowing you to conveniently manipulate the data stored in them. To assign values to elements of an array:

1. Read the values in from a file on the data cassette.
2. Use the READ command to read values from DATA statements.
3. Code an assignment statement for each element in the array. This is practical for short arrays.

Program example:

This program uses a string array to store the names of the days of week. When you enter a number from one to seven, the program displays the name of day.

```
10 REM dim
100 DIM day$(7)
200 ONERR GOTO 2100
1000 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY
1100 DATA THURSDAY,FRIDAY,SATURDAY
1200 FOR day = 1 TO 7
1300 READ day$(day)
1400 PRINT day; TAB(5); day$(day)
```



```

1500 NEXT day
1600 INPUT "Enter 1 to 7>"; day$
1700 IF day$ < "1" OR day$ > "7" THEN PRINT "WRONG": GOTO 1600
1800 subscript% = VAL(day$)
1900 PRINT "DAY="; day$(subscript%)
2000 GOTO 1600
2100 LIST: REM when you type ctrl-c and press return key

```

Helpful hints:

You cannot redimension an array, so you must define an array with the maximum size you will need to store your data.

Arrays take up space in memory. Plan your requirement; if you don't have enough room in memory, store part or all of the data as a direct-access file on cassette.

If you don't dimension an array and use a name with a subscript, BASIC automatically defines an array with eleven elements—from zero to ten.

DRAW shape number AT column,row

Use either form of the DRAW statement to place a shape that you design yourself on the high-resolution screen. Use a number or arithmetic expression to define the *shape number*, *column* and *row* required by the DRAW statement.

Immediate-mode example:

This example illustrates the two forms of DRAW using the default shape, 1, with the color orange(5).

```

You type      HGR
ADAM displays black screen
You type      HCOLOR=5
You type      ROT=0
You type      SCALE=5
You type      DRAW 1 AT 50,100

```

ADAM displays a square with a line coming down from its midpoint at row 50, column 100

```

You type      DRAW 1

```

ADAM displays another square, beginning at the point where BASIC finished DRAWing the first shape.

```

You type      HPLOT 150,50
You type      DRAW 1

```

ADAM draws a third square with the beginning midpoint at column 150, row 50.

Program example:

This program allows you to use hand controller 1 to rotate and change the size of the shape. Pushing the joystick left or right rotates the shape, while pressing the keypad sets the scale or size of the shape.

```

10 REM draw
900 ONERR GOTO 3000
1000 HGR
1100 HCOLOR = 5
1200 big = 1: SCALE = big
1500 DRAW 1 AT 110, 80
2000 REM draw loop
2050 XDRAW 1 AT 110, 80
2100 rotation = PDL(3)
2200 size = PDL(13)
2300 ROT = rotation
2400 IF size <> 15 THEN SCALE = size: big = size
2450 PRINT "scale="; big; " rot="; rotation
2500 DRAW 1 AT 110, 80
2900 GOTO 2000
3000 TEXT

```

Helpful hints:

Appendix D explains how to define a shape, create a shape table and use DRAW and other high-resolution commands. The following summary gives you a checklist of what you must do to draw shapes.

- 1) Create a shape table.
- 2) Load the shape table into memory from a cassette file using the BLOAD statement or POKE it into memory by READING numeric values from DATA statements.
- 3) POKE the memory address of the shape table into decimal memory locations 16766 and 16767.
- 4) Use the HCOLOR statement to define the color. A shape can have only one color.
- 5) Use the ROT statement to define the orientation of the shape on the screen. Maybe you want it upside down.
- 6) Use the SCALE statement to define the size of the shape. Maybe you want it twice as large.
- 7) Use HGR to display high-resolution memory on the video screen.
- 8) If you use the DRAW command without the row and column then you can use the HPLOT statement to define the row and column where BASIC begins to draw the shape.
- 9) Use DRAW to paint the picture on the screen.
- 10) Use XDRAW to remove the picture from the screen.

END

Use the **END** statement to terminate the program and return to the command mode. If you want to terminate the program after the last statement in the program, you can omit the **END** statement. If you are using a structured program, you should place an **END** statement at the end of the main routine.

Program example:

The main process in the following program executes two subroutines. After printing a message, each subroutine returns to the main process. The **END** statement terminates program execution, preventing subroutine A from executing again. You may want to type the **TRACE** command before **RUN**ning this program, so you can see the execution sequence. Use **NO-TRACE** to turn off the line number display.

```
5 REM a-end
10 REM end
1000 REM main process
1100 GOSUB 2000
1200 GOSUB 3000
1900 END
2000 REM subroutine A
2100 PRINT "subroutine A"
2990 RETURN
3000 REM subroutine B
3100 PRINT "subroutine B"
3990 RETURN
```

ERRNUM (O)

You use this function in your **ONERR** error-handling routine to determine what error occurred. It returns a number between 0 and 255 that identifies the error. When an error occurs, SmartBASIC normally terminates the program and displays an error message. However, errors caused by bad input in your program may be able to bypass or request new input.

Immediate-mode example:

If you try to enter a number greater than 32767 into an integer variable, BASIC will display the error message ?Illegal Quantity Error which has the **ERRNUM(O)** of 53.

You type **INPUT "ENTER INTEGER>";I%**
ADAM displays **ENTER INTEGER>**


```

You type      40000
ADAM displays ?Illegal Quantity Error
You type      ?ERRNUM(O)
ADAM displays 53

```

Program example:

This program shows how to use ONERR, ERRNUM(O) and CLRERR. The ONERR command at 1000 tells BASIC to execute line number 9000 when an error occurs, rather than terminate the program. At 9000 the value returned by the ERRNUM(O) function is compared to the Illegal Quantity Error number. If equal then the program displays an error message to the operator and then GOTOs the INPUT command to give the operator another chance to enter the data correctly.

The CLRERR command turns off the ONERR command, and BASIC reverts to the normal error-handling procedures of terminating the program and displaying an error message. The statement at 1400 forces an error to show that the CLRERR worked.

If you terminate the program at the INPUT statement by typing CONTROL-C and RETURN, the ONERR routine prints the value returned by the ERRNUM function when you type CONTROL-C, number 255.

```

10 REM  errnum
1000 ONERR GOTO 9000
1200 INPUT "ENTER INTEGER>"; i%
1300 CLRERR
1400 i% = 40000
9000 IF ERRNUM(O) <> 53 THEN 9100
9050 PRINT "only 32767 to -32767 valid"
9060 GOTO 1200
9100 PRINT "ERRNUM(O)="; ERRNUM(O)
9200 END

```

Helpful hints:

The value returned by ERRNUM(O) stays the same, until replaced by another error.

Refer to ONERR for another example of error handling.

EXP (arithmetic expression)

Use the EXPonentiation function to compute the value of the constant e , 2.718289, raised to the power of the value of the arithmetic expression.

EXP (arithmetic express)

Use the EXPonentiation function to compute the value of the natural number the constant e , 2.71828183, raised to the power of the value of the arithmetic expression.

Immediate-mode example:

Any number raised to the first power is the number. To display the value for e

You type ?exp (1)
ADAM displays 2.71828183

Program example:

This program prints 17 values of the function from -4 to +4 adding .5 after each FOR . . . NEXT loop.

```

5 REM a-exp
10 REM exp
500 PRINT "EXPONENTIAL FUNCTION"
1000 FOR power = -4 TO 4 STEP .5
1100 PRINT "EXP("; power; ")="; TAB(15); EXP(power)
1900 NEXT power

```

Helpful hints:

See the explanation of the LOG function.

FLASH

When you want to catch the operator's attention, you use the FLASH command to blink anything displayed by the PRINT command. FLASH works in the TEXT mode but *not* in the TEXT area of the GR and HGR modes. BASIC accomplishes the blinking by rapidly switching back and forth between the NORMAL white on black and INVERSE black on white display mode.

Immediate-mode example:

To FLASH the words "EXECUTING PROGRAM" in the middle of the screen

You type TEXT
You type FLASH
You type HTAB 5:VTAB 12:?"EXECUTING PROGRAM"

To turn the blinking off

You type NORMAL
You type HTAB 5:VTAB 12:?"EXECUTING PROGRAM"

The NORMAL command doesn't turn off the text already flashing. You must rePRINT the data to stop it from blinking.

Program example:

The FLASH program allows you to set any of the three TEXT display modes by pressing one of the roman numeral function keys. It prompts you to enter a character string, which it then displays in the mode you've selected. Pressing CTL-C terminates the program.

```

10 REM flash
1000 NORMAL: HTAB 1: VTAB 23
1100 PRINT " I "; " II "; " III "
1200 PRINT "NORML"; "inver"; "FLASH"
1300 PRINT "PRESS FUNCTION KEY"
2000 GET key$: key% = ASC(key$)
2050 IF key% = 3 THEN END
2100 IF key% = 129 THEN NORMAL
2200 IF key% = 130 THEN INVERSE
2300 IF key% = 131 THEN FLASH
2400 INPUT "TEXT>"; sentence$
2500 HOME: VTAB 10: PRINT sentence$
2600 GOTO 1000

```

Helpful hints:

Use FLASH only to get the operator to respond. The blinking is annoying, so once the program gets the response, turn off the blinking by executing the NORMAL command and rePRINTing the text at the screen location FLASHing.

The TEXT command cancels the FLASH command, so you must execute it again if you want subsequent PRINT text to blink.

Making a beeping sound by PRINTing CHR\$(4) is another way to attract attention.

FOR counter-initial value TO final value STEP increment

Frequently, you need to repeat a group of statements a certain number of times. The FOR and the NEXT statement used together allow you to define a loop more efficient in execution time and memory space than an IF and GOTO combination.

The *counter* must be a real variable; an integer will not work.

The *initial value* can be a single number, often one, or any arithmetic expression. You can use a positive or negative value.

The *final value* can be a single number, often the number of times you want to repeat the loop, or an arithmetic expression having either a positive or a negative value.

The *increment* and the word *step* are optional. If omitted, BASIC assumes one as the increment. If present, the increment must be an arithmetic expression having either positive or negative value.

How it works:

Consider two FOR . . . NEXT loops. One counts up from one to two, and the other counts down from two to one.

```
10 REM FOR UP
100 FIRST=1:LAST=2
1000 FOR COUNTER=FIRST TO LAST
1100 PRINT "COUNT=";
1200 NEXT COUNTER
```

The first time BASIC executes the FOR statement it assigns the real variable *count* to the *initial value* stored in variable *first*. Since *first* has a value one, *counter* now has a value one. BASIC then executes all statements until it reaches the NEXT statement. So that BASIC can match up the FOR and NEXT, both must have the same real variable counter name—in this case the variable name *counter*. The loop has only one statement—the PRINT statement, which displays the value of the *counter* so you can see the flow of execution.

At the NEXT statement BASIC adds the STEP *increment* to the *count*. In this case, since the FOR statement omits the STEP value, BASIC uses one as the default increment. It adds one to *counter* giving it a value of two. Then it compares the value of *count* with the *final value*. In this case, it compares the value in *counter*, two, with the value in *last*, two.

With a positive *increment* and:

- $count \leq final\ value$, BASIC transfers control to the statement following the FOR. In this case line 1100 the PRINT statement.

- $count > final\ value$, the loop terminates and the statement following the NEXT will execute. After the second loop COUNTER becomes 3, which is greater than 2 so the loop and program terminate.

Now change the program to go backwards from 2 to 1.

```
100 FIRST=2:LAST=1
1000 FOR COUNTER=FIRST TO LAST STEP -1
```

The negative STEP value, minus one, causes COUNTER to go down in value from 2 to 1 and then 1 to 0.

With a negative *increment*, and

- $count < final\ value$, BASIC transfers control to the statement following the FOR.

- $count \geq final\ value$, then statement following the NEXT statement executes. When COUNTER becomes zero, which is less than the 1 in LAST, the loop terminates.

RUN both programs and watch how they work.

RUN the negative FOR . . . NEXT

Immediate-mode example:

To use the FOR . . . NEXT loop in the immediate mode you must type all the statements on one line. This example shows a useful debugging technique when you want to list the contents of an array. Here the array has all zeros.

You type FOR sub=1 TO 3: ? sub; TAB(4); a(sub); NEXT sub
 ADAM displays 1 0
 2 0
 3 0

Program example:

You can nest FOR . . . NEXT loops, in other words having a loop within a loop. This program loads a two-dimensional array and prints each element. The outer loop processes the rows, while the inner loop processes the columns within each row.

```

10 REM   for
100 DIM array%(4, 3)
1000 DATA  3,4,4,5
1100 DATA  3,0,0,5
1200 DATA  3,4,4,5
1900 GR
2000 REM   outer loop
2100 FOR row = 1 TO 3
2200 REM   inner loop
2300 FOR column = 1 TO 4
2400 READ hue
2500 array%(column, row) = hue
2600 COLOR = hue
2700 PLOT column, row
2800 NEXT column
2900 NEXT row

```

Helpful hints:

A FOR . . . NEXT loop always executes once because the comparison takes place at the NEXT statement.

While you can branch out of loops with a GOTO, use this technique carefully or your program may become confusing. A loop that starts at the top and ends at the bottom is easier to follow than one that, in the middle, transfers execution out of the loop to someplace else in the program.

FRE(0)

Use this function to find out how much memory you have left free for more program lines and data. The function does not use the number inside the parentheses, but you must provide some value. The *Companion* recommends zero.

Immediate-mode example:

You type PRINT FRE(0)

ADAM displays 26097

This PRINT FRE(0) was done immediately after typing NEW. It means that 26097 bytes is the maximum memory you have available for your program and data. The BASIC interpreter and the OS-7 operating system take up the rest of memory. The maximum memory may change with later releases of BASIC and the operating system.

Program example:

The following program shows you how to use the FRE function to determine how much space your data and program take up. It assigns the character string constant "0123456789" to 10 elements in the array *array\$*. Notice that when you execute the DIM, BASIC allocates space for empty or null entries in the array. When the program stores a 10-character string in an array element, it reduces free memory by 13 bytes, three bytes for control information and 10 for the characters in the string constant.

```
5 REM a-fre
10 REM fre
50 PRINT "before dim="; FRE(0)
100 DIM array$(10)
150 PRINT "after DIM="; FRE(0)
1000 FOR sub = 1 TO 10
1100 array$(sub) = "0123456789"
1200 PRINT FRE(0)
1300 NEXT sub
```

Helpful hints:

Use the FRE to determine if your program is running out of available memory space. Place it after the initialization phase, when your program has loaded all arrays.

GET string\$

To get a single character from the keyboard without having to wait for the operator to press the RETURN key, use the GET statement. GET does not echo the character typed to the keyboard, which makes it great for entering passwords because the person looking over your shoulder doesn't see the letters you type on the video screen.

Immediate-mode example:

The following FOR . . . NEXT loop allows you to enter 10 characters, and uses the PRINT command to display each on a separate line.

You type FOR C=1 to 10:GET key\$?:key\$:next c
ADAM displays the 10 characters you typed.

Program example:

With the INPUT statement you can't control the number of characters typed into a string field, and it displays every character you type on the video screen. Since GET doesn't display the character typed, it's useful for entering passwords—the person looking over your shoulder doesn't see what you typed.

This program allows you to enter five capital letters. Pressing the RETURN key terminates the routine, but after the five characters are entered the subroutine terminates automatically, without pressing the RETURN key. The test for CONTROL-C provides a universal exit so the program does not loop endlessly if you don't enter five capital letters in succession.

```

10 REM  get input password
1000 size% = 5
1010 true% = 1
1020 GOSUB 2000
1040 PRINT
1050 PRINT "password="; password$
1090 END
2000 password$ = ""
2050 TEXT
2060 IF error% THEN PRINT CHR$(7); "NOT A-Z. Start again"
2070 error% = 0
2080 PRINT "enter password=";
2100 GET key$
2110 IF key$ = CHR$(3) THEN END
2120 IF key$ = CHR$(13) THEN RETURN: REM return key
2140 IF key$ >= "A" AND key$ <= "Z" THEN 2160
2150 error% = true%
2155 GOTO 2000
2160 password$ = password$+key$
2180 IF LEN(password$) = size% THEN RETURN
2190 GOTO 2100

```

Helpful hints:

The GET has no particular use in the immediate mode.

Chapter 7 uses the GET statement in the picture-maker program.

See ON . . . GOSUB for another example of GET.

GOSUB.line number

You use this statement to transfer statement execution to a subroutine beginning at the line number. The RETURN statement in the subroutine transfers execution back to the statement following the GOSUB. You can nest GOSUB statements. This means that the subroutine can invoke another subroutine using a GOSUB. BASIC stacks the return line number in memory. Then as it executes each RETURN or POP, it removes the top line number from the stack. For a RETURN statement it transfers statement execution to the line number just removed. You *cannot* use a variable with the line number stored in it. The actual line number must follow the word GOSUB.

Immediate-mode example:

Not recommended.

Program example:

If you have read Chapter 5 you know that most processes break down into smaller subprocesses, which in turn may break down into even simpler subprocesses. You should follow good program-design technique and use GOSUBs to show, in the beginning of the program, the overall processes performed by the program. Each GOSUB evokes a subroutine that performs a subprocess. The following illustrates these designs with nested GOSUBs.

```
10 REM gosub
1000 REM main process
1100 GOSUB 8000: REM initialization
1200 GOSUB 2000: REM input
1300 GOSUB 3000: REM calculation
1400 GOSUB 4000: REM output
1600 END
2000 REM initialization
2100 PRINT "initialization"
2900 RETURN
3000 REM calculation
3100 GOSUB 3500
3190 RETURN
3300 PRINT "first calculation"
3390 RETURN
3500 PRINT "second calculation"
3590 RETURN: REM exit second calculation
4000 RETURN: REM dummy output
5000 RETURN: REM dummy termination
8000 RETURN: REM dummy initialization
```

Helpful hints:

If the line number in the GOSUB does not exist in the program, you will get the following error message:

?Undefined Statement Error

See POP and RETURN.

ON . . . GOSUB allows you to select one of several subroutines to execute.

GOTO line number

When you want to transfer control to a particular program line, you use the GOTO statement. Unlike the GOSUB, BASIC *does not* store a return line in the stack. The GOTO allows you:

1. To create a loop. Transfer backward to a lower line number, repeating a group of statements until a termination condition occurs.
2. To transfer to a particular statement depending on a condition, when used with the IF condition test statement.
3. To skip around a group of statements that you don't want to execute.

Immediate-mode example:

Not usually used in immediate mode, but when you're testing a program, instead of typing RUN, which starts execution at the beginning of your program, you could type GOTO *line number* to start execution someplace in the middle of the program. Be careful that variables get correctly initialized. You can also use it after a STOP statement.

Enter this two statement program

```
You type      new
You type      1000 ?"line 1000"
You type      2000 ?"line 2000"
You type      RUN
ADAM displays line 1000
               line 2000
```

The RUN started program execution with the first statement at line number 1000. Now use the GOTO statement to start execution at line number 2000.

```
You type      GOTO 2000
ADAM displays line 2000
```

BASIC started execution with line number 2000.

Program example:

This program illustrates the three ways you can use the GOTO.

1. Creating a loop. The GOTO at line number 1990 loops back to a lower line number, in this case 1000, to repeat the processing cycle.
2. Transfer depending on condition. The GOTOs as part of the IF commands at 1100, 1200 execute when the condition *truth\$* = "1" or *truth* = "0" becomes true. They transfer execution to the PRINT statement at 1800.

3. Skip around statements. After displaying the error message, the error routine skips around the PRINT "thank you" statement by executing a GOTO 1900 which instructs the operator to "Please do it again."

```

10 REM goto
300 beep$ = CHR$(7)+CHR$(7)
1000 INPUT "ENTER 1 OR 0?"; truth$
1100 IF truth$ = "1" THEN PRINT "TRUE": GOTO 1800
1200 IF truth$ = "0" THEN PRINT "FALSE": GOTO 1800
1300 PRINT "ONLY 1 or 0 VALID"; beep$
1400 GOTO 1900
1800 PRINT "Thank you for following instructions"
1900 PRINT "Please do it again"
1990 GOTO 1000

```

Helpful hints:

Where possible use the GOSUB and FOR . . . NEXT loops. Excessive use of GOTO statements will make the logic flow of the program difficult to follow. A GOSUB statement should always come back to the following statement. The logic flow after a GOTO may lead you through a complicated maze without a road map.

GR

Use the GR command to change the video display mode to the low-resolution graphics display mode. In this mode you can plot colored squares on a grid 40 columns by 40 rows. Rows and columns are numbered from 0 to 39. Four lines on the bottom allow you to PRINT instructions in TEXT mode. The GR command sets the low-resolution COLOR=0 (black) and clears the screen to black.

Immediate-mode example:

```

You type      GR
You type      COLOR=15 (white)
You type      HLIN 20,25 AT 39
ADAM draws white line at bottom middle of screen.
You type      TEXT
ADAM switches you back into TEXT mode, and the white line disappears.
The [ prompt appears in the top left corner.

```


Program example:

The picture maker does not support drawing a color background after you clear the screen. You may want to incorporate part of the following routine into your version of the picture maker to give you this feature.

```

10 REM  gr color background
1000 GR
1050 FOR hue = 1 TO 15: REM  zero is black
1060 COLOR = hue
1100 FOR row = 0 TO 39
1200 HLIN 0, 39 AT row
1300 NEXT row
1400 NEXT hue
1500 TEXT

```

Watch the screen turn different colors.

Helpful hints:

Chapter 7 explains how to use these statements and how to create a low-resolution picture-maker program.

Remember, execute GR before setting the desired color because GR sets the color to black.

See COLOR for the 16 colors you can use.

PLOT draws a square at a given column and row.

HLIN draws a horizontal line from one column to another column in a given row.

VLIN draws a vertical line from one row to another row in a given column.

HCOLOR=color number

You use the HCOLOR statement to set the color displayed by the BASIC high-resolution graphics statements HPILOT and XDRAW. The table below lists the names and numbers of the 16 colors.

<i>Number</i>	<i>Color</i>	<i>Number</i>	<i>Color</i>
0	Black-1	8	Brown
1	Green	9	Dark blue
2	Violet	10	Gray
3	White-1	11	Pink
4	Black-2	12	Dark green
5	Orange	13	Yellow
6	Blue	14	Aqua
7	White-2	15	Magenta

Immediate-mode example:

These statements draw a yellow square.

You type HGR

You type HCOLOR=13

You type HPLOT 90,96 TO 90,103 TO 97,103 TO 97,96 TO 90,96

ADAM displays a little square box

Program example:

The program uses the HPLOT to paint the screen with 15 vertical bars for 15 of the 16 high-resolution colors. Since the screen begins black the bar for it, the value zero, is not painted with the HPLOT.

```

10 REM    hcolor
1000 HGR
1100 hue = 0
1200 FOR column = 8 TO 128 STEP 8
1300 HCOLOR = hue
1310 HOME: PRINT "HCOLOR="; hue
1350 FOR line = column TO column+5
1400 HPLOT line, 0 TO line, 159
1450 NEXT line
1600 hue = hue+1
1700 NEXT column
1800 HOME
1900 PRINT "0123456789111111"
2100 PRINT "          012345"
2200 GET key$
2300 IF ASC(key$) = 3 THEN END
2400 HOME
2500 PRINT "BGVWBOBWBDGPDYAM"
2510 PRINT "LRIHLRLHRBRIGEQA"
2520 PRINT "AEOIAAUIOLENRLUG"
2530 PRINT "CELCNETWUYKELAE";
2700 GET key$
2800 IF ASC(key$) <> 3 THEN 1800
2900 END

```

Helpful hints:

Refer to HPLOT and HGR for other examples for using HCOLOR.

HGR

Use this command when you want to change the video display mode to high-resolution graphics mode. In this mode you can plot colored dots on a

grid 256 columns by 160 rows. Columns are numbered left to right from 0 to 255. Rows are numbered top to bottom from 0 to 159. Four lines on the bottom allow you to PRINT instructions in TEXT mode.

Immediate-mode example:

To demonstrate the boundaries of the high-resolution screen, type these commands.

You type HGR

You type HCOLOR=5

You type HPLOT 0,0 TO 255,0 TO 255,159 TO 0,159 TO 0,0

ADAM draws a box around the edge of the video screen starting from the left top. It draws an orange line

from left top across to right top,
then down to right bottom,
then across the bottom to left bottom,
and finally up to the starting location left top (0,0).

WARNING: Many televisions overscan the screen, resulting in loss of picture at the four edges. If you don't see the above border, try drawing a smaller box. For example,

You type HPLOT 8,8 TO 247,8 TO 247,159 TO 8,159 TO 8,8

Program example:

You may have seen Coleco's changing color triangle in demonstrations of ADAM. This slight variation uses 15 colors, but not black (0). The HPLOT statement at 2300 draws lines from row 190 to the apex at column 125, row 1. Notice that even though one color is specified, different colors appear.

```

10 REM    hgr color triangle
1000 HGR
1100 hue = 0
2000 FOR hue = 1 TO 15
2100 HCOLOR = hue
2150 IF hue = 4 THEN 2500
2170 HOME: PRINT "hcolor="; hue
2200 FOR column = 1 TO 255 STEP 5
2300 HPLOT column, 190 TO 125, 1
2400 NEXT column
2500 NEXT hue

```

Helpful hints:

Remember to execute HGR before setting HCOLOR because HGR sets HCOLOR=0 (black) overriding any previous HCOLOR statement.

TEXT returns you to the TEXT mode.

GR puts you in the low-resolution graphics mode.

HGR2 gives you a full screen of 192 rows by eliminating four text lines displayed in HGR mode.

HPLOT draws dots and lines.

DRAW places a shape on the screen, but first you need to create or load a shape table. Refer to Appendix D for instructions on how to create a shape-table program.

XDRAW, ROT and SCALE are other high-resolution commands you can use.

HGR2

You use this mode when you want to display the full screen as high-resolution dots. This command gives you a full 192 rows by eliminating four text lines at the bottom of the screen.

Immediate-mode example:

To see what the HGR2 command does,

You type HGR2

ADAM displays a black screen.

To get back into TEXT mode,

You type TEXT

BASIC doesn't display the letters TEXT but does execute the command when you press the RETURN key.

Program example:

See HGR.

Helpful hints:

See HGR.

HIMEM: (decimal address)

Sets the highest memory location that BASIC can use to store data. While this is a valid command, the *Companion* recommends you do not use it. To reserve space for your shape table or machine-language program, use the LOMEM: command.

HLIN from column, to column AT row

In low-resolution graphics mode (GR) you use HLIN to draw a thick, colored, horizontal, left to right line in a particular row. The last COLOR command BASIC executed determines the color of the line. Remember that GR sets COLOR=0 (black). A black line appears invisible against a black

background. Therefore, always assign COLOR to a visible COLOR value (1 to 15) after doing a GR before executing the HLIN.

A grid of squares consisting of 40 columns and 40 rows forms the low-resolution graphics screen. Columns are numbered left to right from 0 to 39. Rows are numbered top to bottom from 0 to 39. If either the column or row is greater than 39, SmartBASIC displays the error message:

?ILLEGAL QUANTITY

You can define the *from column*, *to column* and *row* with a number, numerical variable or arithmetic expression.

Immediate-mode example:

To illustrate the width of the low-resolution screen, the following commands draw a brown line on the top row of the grid.

```
You type      GR
You type      COLOR=3
You type      HLIN 0, 0 AT 0
You type      HLIN 0, 39 AT 0
```

Program example:

The following program uses the random function to draw random-length horizontal lines, in random rows and in random colors.

```
10 REM hlin
1000 GR
2000 begcol% = RND(1)*39
2100 endcol% = RND(1)*39
2200 row% = RND(1)*39
2300 hue% = RND(1)*15
2400 COLOR = hue%
2500 HLIN begcol%, endcol% AT row%
2900 GOTO 2000
```

Helpful hints:

While SmartBASIC will allow you to define the columns in reverse order—that is, *to column*, *from column*—you should avoid this practice because it will confuse anyone, even yourself, who tries to understand what the program does. For example,

both
HLIN 5, 10 AT 10 RECOMMENDED

and
HLIN 10, 5 AT 10 NOT RECOMMENDED

draw the same line from column 5 to column 10 at row 10, but the

Companion recommends you use the first statement.

VLINE draws a vertical line in a specified column.

PLOT draws a colored square at a particular column and row.

HOME

When you want to clear the text area of the screen, you use this command. HOME positions the cursor to the top line. In the TEXT mode, this command clears all 31 lines and positions the cursor at the top of the screen just to the right of the | prompt. If you're in either the low- or high-resolution graphics screen modes, it clears the four text lines at the bottom of the screen and positions the cursor at the top of the text area.

Immediate-mode example:

Type HOME to give yourself a full screen of 24 lines to write your program.

You type NEW

You type HOME

The NEW command clears BASIC memory, and HOME clears the video screen.

Program example:

This program shows how HOME works in each video mode that displays text—TEXT, GR or HGR. The delay loop allows you to see the message; otherwise, because BASIC executes commands so fast, you wouldn't see the message.

```
10 REM home
1000 TEXT: PRINT "TEXT"
1100 GOSUB 3000: REM delay
1120 HOME: PRINT "TEXT HOME"
1130 GOSUB 3000: REM delay
1140 GR: PRINT "GR"
1150 GOSUB 3000: REM delay
1160 HOME: PRINT "GR HOME"
1170 GOSUB 3000
1180 HGR: PRINT "HGR"
1190 GOSUB 3000
1200 HOME: PRINT "HGR HOME"
1210 GOSUB 3000
1220 GOTO 1000
3000 REM delay
3100 FOR i = 1 TO 2000: NEXT: RETURN
```


Helpful hints:

The TEXT command automatically does a HOME.

HPlot column,row TO column,row . . .

When you want to draw colored dots or lines in the high-resolution mode, you use this command. The *column* value ranges from 0 on the left to 255 on the right. The *row* value ranges from 0 at the top of the screen to either 159 in HGR mode or 191 in HGR2 mode.

Immediate-mode example:

To plot a single dark-green dot at column 128, row 80,

You type HGR

You type HCOLOR=12

You type HPLOT 128,80

ADAM displays a dot in the middle of the screen

To draw a vertical line up and down,

You type HPLOT 124,74 TO 124,86

ADAM displays a line from row 74 to row 86 in column 124

To draw a horizontal line from left to right,

You type HPLOT 124,74 TO 132,74

ADAM displays a line from column 124 to column 132 in row 74.

You type HPLOT 124,74 to 132,86

ADAM displays a line from column 124, row 74 to column 132, row 86

Instead of writing three statements, you could have written an HPLOT statement to draw the triangle.

You type HGR

You type HCOLOR=12

You type HPLOT 124,74 to 124,86 to 132,86 to 124,74

ADAM displays a triangle

Program example:

Rather than go to the time and trouble of defining a shape for DRAW to display, you can use HPLOT to create the figure or shape. The following program illustrates how to do animation. The subroutine beginning at 3000 uses HPLOT statements to sketch a face in a square box. The FOR . . . NEXT loop sets HCOLOR to orange and draws the face. By setting the color to black and sketching the face again, you erase the face. You repeat the draw and erase cycle 16 columns to the right. By repeatedly erasing and drawing at a different location a short distance away, the program creates the illusion of fluid movement or animation, causing the face to move across the screen from left to right.


```

10 REM restore
500 true$ = 1
1000 GR
1100 HTAB 1: VTAB 24: PRINT "ENTER end TO QUIT";
3000 REM get color name
3010 HTAB 1: VTAB 22: PRINT SPC(31);
3020 HTAB 1: VTAB 22
3030 INPUT "enter color="; shade$
3040 IF shade$ = "end" THEN TEXT: END
3100 REM      search for color name
3105 RESTORE
3110 found$ = false$
3120 FOR hue = 0 TO 15
3125 READ rainbow$
3130 IF shade$ = rainbow$ THEN GOSUB 4000
3140 NEXT hue
3150 IF NOT found$ THEN VTAB 21: HTAB 1: PRINT "BAD COLOR NAM
3190 GOTO 3000
4000 REM draw vertical line
4050 HTAB 1: VTAB 21: PRINT SPC(31);
4100 found$ = true$
4300 COLOR = hue
4400 VLIN 0, 39 AT 20
4900 RETURN
9000 DATA      "black"
9010 DATA      "magenta"
9020 DATA      "darkblue"
9030 DATA      "darkred"
9040 DATA      "darkgreen"
9050 DATA      "grey-1"
9060 DATA      "green"
9070 DATA      "blue"
9080 DATA      "yellow"
9090 DATA      "red"
9100 DATA      "grey"
9110 DATA      "pink"
9120 DATA      "palegreen"
9130 DATA      "tan"
9140 DATA      "cyan"
9150 DATA      "white"

```

Helpful hints:

HGR must precede HPLOT because BASIC clears the screen.

To remove a figure from the high-resolution screen, set HCOLOR=0 (black) and HPLOT the figure again.

Use DRAW and XDRAW to display more complex shapes.

HTAB column

In the TEXT display mode, when you want to move the cursor to a particular column in the current line, you use this command. Text columns range from 1 at the left of the screen to 31 at the left.

Immediate-mode example:

To position the cursor to column 15 in the current row,

You type TEXT

You type HTAB 15:15

ADAM displays 15 on line 2, columns 15 and 16

Program example:

The FOR . . . NEXT loop in this program uses the HTAB to print a diagonal line of 22 numbers. Each number begins in the column the number specifies.

```
10 REM htab
500 PR #1
1000 TEXT
1100 FOR line = 1 TO 22
1200 column = line
1300 HTAB column: PRINT line
1400 NEXT line
```

Helpful hints:

VTAB positions the cursor to a particular row. Usually you pair the HTAB and VTAB commands to position the cursor to a *column, row* location.

HTAB works only on the screen and doesn't affect the actual printed output. Use the PRINT functions TAB and SPC to position data within a print line.

IF condition value THEN command: command: . . .

Use this command when you want to test the truth of a condition to determine what statements to execute. The *condition value* can be zero, which means false, or nonzero, which means true. On a true condition, BASIC executes the statements after the THEN.

The condition value may be:

An integer or decimal variable like *switch%*.

An arithmetic expression like *button% + joystick%*.

A relational expression like *amount% > limit%*.

A compound logical expression like *switch% AND button% + joystick%*.

A value of a function like *SGN (amount%)*.

The NOT logical operator reverses the truth of a condition. For example, while *1<2* is true, *NOT 1<2* is false.

The statements executed on a true condition can be:

One command like PRINT "TRUE."

Two or more commands separated by colons (:) like PRINT "TRUE":
GOTO 5000.

A line number like 5000, which is a shorthand way of writing GOTO 5000.

This works only if it's the sole true command.

Immediate-mode example:

You type IF true% THEN ?"TRUE"

ADAM displays nothing, just the prompt].

When BASIC first allocates space for an integer or decimal variable, it assigns the value zero to the variable. Zero means false, so the word TRUE doesn't print.

Now make true% really true by assigning a nonzero value.

You type true%=1

You type IF true% THEN ?"TRUE"

ADAM displays TRUE

A nonzero value like one gives a true result.

Program example:

The aim of this program is to show you various ways of using the IF command. It allows you to enter two string variables, and then it makes several tests and display messages describing the results of the tests. Experiment with different numbers. If you don't enter a valid number, the program will not execute the numeric tests.

```

10 REM if
1000 INPUT "A="; a$
1100 INPUT "B="; b$
1200 IF a$ = "end" OR b$ = "end" THEN END
1300 IF LEN(a$) THEN a = VAL(a$)
1400 IF LEN(b$) THEN b = VAL(b$)
2000 IF a$ = b$ THEN PRINT "A$=B$"
2100 IF a$ < b$ THEN PRINT "a$<b$"
2200 IF a$ > b$ THEN PRINT "a$>b$"
2500 IF NOT (a+b) THEN 1000
3000 IF (a-b*2) = 4 THEN PRINT "a-b*2=4"
3100 IF a > b THEN PRINT "a>b"
3200 IF NOT (a-b) THEN PRINT "a-b end": ENI
3300 IF a-b THEN 1000

```

Helpful hints:

Refer to the explanations of the logical operators NOT, OR and AND for further examples of the IF command.

INIT volume name

This immediate-mode command allows you to zero out the catalog on the cassette. WARNING!! Once you zero out the directory, you lose all access to files on the tape. In effect you *deleted* all the files.

The first time you use a cassette, you should INIT the tape assigning a volume name that appears on the top line of the BASIC catalog display. Use a volume name of 1 to 10 characters in length. If you type more than 10 characters, ADAM will store only the 10 left-hand characters.

Immediate-mode example:

If you want to type the programs in this book, place a blank cassette in the tape drive.

```
You type      CATALOG
You type      INIT COMPANION
You type      CATALOG
```

The cassette tape motor sounds, and the catalog screen appears on the video screen.

Helpful hints:

Always use CATALOG command before INITIALIZING a directory to make sure you don't need any files on the tape.

INPUT "prompt";variable name, variable name . . .

When your program needs INPUT from the keyboard or tape cassette, you use this command. When you provide a list of variables, each field entered must be separated by a comma.

For keyboard input, you have the option of not using the prompt character string, and BASIC will use a question mark to indicate that it's waiting for INPUT. Unless you have already displayed a prompt with the PRINT statement, the *Companion* recommends that you always provide a meaningful prompt to describe the data expected by the program.

For tape cassette input, you must provide a null string prompt "(no characters between the string delimiters)" to prevent BASIC from displaying a question mark (?) on the screen as it reads each record from the tape. Before executing the INPUT command, you must have OPENed the file and executed the READ command. Refer to READ and OPEN for examples using the INPUT command to get data from tape.

Immediate-mode example:

Suppose you want to input a row and column to position the cursor.

```
You type      HOME
You type      INPUT row, column
ADAM displays ?
You type      5,5
You type      HTAB column:VTAB row:?column,row
ADAM displays 55 (at column 5, at row 5)
```

Program example:

This program illustrates the different formats of the INPUT command. When the INPUT command lists two or more input fields, if you only enter one, BASIC prompts you with ? for each of the remaining fields in the list. When you RUN the program, you will realize how important prompts are to creating user-friendly programs.


```

10 REM a-input
1000 INPUT "enter string>"; string$
1100 INPUT "enter integer>"; whole%
1200 INPUT "enter decimal>"; decimal
1300 PRINT string$, whole%, decimal
1400 INPUT a$, b$
1500 INPUT a
1600 PRINT a$, b$, a

```

Helpful hints:

Always use the prompt feature to tell the person using the program what data he/she must enter.

While you can INPUT integer and decimal variables, the *Companion* recommends that in most instances, you input character strings and then validate the data. If the operator made a mistake, display an error message. If the data is valid, use the VAL or ASC functions to convert the data to a numeric variable. Also, refer to the *fieldinput* routine explained in Chapter 10.

The GET command allows single character input without having to press the RETURN key.

INT(decimal value)

When you want to eliminate the fractional part of a decimal number and show only the INTeger, use this function.

Immediate-mode example:

Suppose you want to print only the whole-number part of the number 345.567 in *amount*.

You type amount = 345.567

You type ?INT(amount)

ADAM displays 345

The .567 fractional decimal part is not printed, because the INT function chopped it off.

Program example:

Sometimes you may want to print the fractional part of a decimal number. By subtracting the integer value of the decimal number, you get the fractional part. The following program allows you to enter a decimal number; it prints the fractional part.

```

10 REM a-int
1000 INPUT "enter decimal number>"; decimal
2000 fraction = decimal-INT(decimal)
3000 PRINT "fraction="; fraction
4000 GOTO 1000

```


Helpful hints:

Although assigning a decimal variable to an integer variable also eliminates the fractional part, an integer variable only has a value range from 32767 to -32767. Assigning a value outside that range gives an illegal-quantity error message. The INT function doesn't have this restriction. For example, you can PRINT INT(98765.432).

INVERSE

In the TEXT mode, to highlight a field, use the INVERSE command to make the normal white characters on black background appear as black on white. This command does not work in the GR and HGR mode.

Immediate-mode example:

To display the words ENTER AMOUNT as black letters on a white background,

You type TEXT

You type INVERSE:?"ENTER AMOUNT"

ADAM displays the letters in inverse video mode. The NORMAL command changes the text display mode back to white letters on a black background.

You type NORMAL

Program example:

By displaying spaces in inverse video, you indicate to the operator the size of the field to enter. The following program doesn't do anything but accept data, but it illustrates the technique. For each of the two fields, *Amount* and *Code*, you define the column, row and field size. The subroutine at 2000 uses the SPC command to display INVERSE video spaces to indicate the maximum-size field to INPUT.

```

10 REM  inverse
1000 HOME
1100 VTAB 12: PRINT "Amount="
1200 VTAB 14: PRINT "Code="
1300 col = 10: row = 12: size% = 8: GOSUB 2000
1500 col = 10: row = 14: size% = 2: GOSUB 2000
1900 END
2000 REM  input routine
2100 HTAB col: VTAB row: INVERSE: PRINT SPC(size%)
2300 HTAB col: VTAB row: INPUT ""; field$
2400 NORMAL
2900 RETURN

```

Helpful hints:

Use the INVERSE command to highlight important information displayed in the TEXT mode. Don't overdo it or you will spoil the effect.

LEFT\$(string\$,number to extract)

The LEFT\$ command allows you to extract a copy of the left side of a character string. The first argument is the character string you want to extract characters from. The second argument is the number of characters to copy.

Immediate-mode example:

To extract a copy of the left three characters from the character string *name\$* that contains ABCDEFG, type the following:

You type name\$="ABCDEFG"

You type ?LEFT\$(name\$,3)

ADAM displays ABC

The LEFT\$ function selected the left three characters, ABC, from the string ABCDEFG. The contents of *name\$* stay the same.

Program example:

Many times the operator must make a YES or NO response to a question asked by the program. Rather than check for the full word YES or NO, use the LEFT\$ function to select only the left character. This gives the operator the convenience of typing only the first letter. The following program asks the operator if the program should print the report on the daisy-wheel printer.

```

10 REM left
500 HOME
600 beep$ = CHR$(7)+CHR$(7)
1000 INPUT "YES/NO?"; ans$
1100 IF LEFT$(ans$, 1) = "Y" THEN GOTO 2000
1150 IF LEFT$(ans$, 1) = "y" THEN GOTO 2000
1200 IF LEFT$(ans$, 1) = "N" THEN GOTO 3000
1250 IF LEFT$(ans$, 1) = "n" THEN GOTO 3000
1300 PRINT beep$; "only YES or NO valid. Try Again"
1400 GOTO 1000
2000 REM yes
2100 PRINT "YES"
2900 GOTO 1000
3000 REM no
3100 PRINT "NO"
3900 GOTO 1000

```

Helpful hints:

You can fill out a short string to a fixed size by taking the left side of the string concatenated (joined together) with a string of spaces. For example, suppose SP\$ contains ten spaces and you want AS\$ to have a length of six characters. If A\$ contains ABC, the following statement will make sure it contains only six characters.

AS\$=LEFT\$(AS\$+SP\$,6)

BASIC reduces this to

AS\$=LEFT\$("ABC",6)

and then

AS\$="ABC "

The letters ABC plus three spaces equals the desired length of six characters.

Refer to the other string functions RIGHTS, MID\$, STR\$ and LEN.

LEN (string\$)

When you want to know the LENgth of a character string, use this function.

Immediate-mode example:

Suppose you want to know the length of AS, which contains ABCDEF.

You type AS="ABCDEF"

You type ?LEN(AS)

ADAM displays 6

There are six characters in the character string ABCDEF.

You can also get the length of two strings concatenated (joined together).

You type BS="123"

You type ?LEN(AS+BS)

ADAM displays 9

The string AS concatenated with BS gives ABCDEF123, which is nine characters.

Program example:

You will often use the LEN function to check if the operator entered any data. If the operator just pressed the RETURN key without typing any other characters, the input string variable will be a null or empty. The LEN function returns a value zero for a null or empty string. The following program requires the operator to enter a filename.

```

10 REM len
500 beep$ = CHR$(7)+CHR$(7)
1000 INPUT "FILENAME>"; filename$
1100 IF LEN(filename$) = 0 THEN 2000
1200 PRINT "FILENAME="; filename$
1300 GOTO 1000
2000 PRINT beep$; "NO FILENAME ENTERED!"
2100 GOTO 1000

```

Helpful hints:

The LEN function always returns an integer value ranging from zero for a null string to 255 for the maximum string length.

LET (variable name = value)

The LET command is the assignment command, but since the word LET is optional, nobody ever uses it.

BASIC has three types of *variable* names:

1. Decimal variable names, which store decimal values like 234.567—for example, *amount* = 234.556.
2. Integer variable names, which store integer or whole-number values from -32767 to 32767—for example, *amount%* = 32767.
3. String variable names, which store character strings from 0 to 255 characters long—for example, *amount\$* = "234.556."

Immediate-mode example:

One choice is:

You type LET A=1

But almost everybody chooses:

You type A=1

The LET is understood.

Helpful hints:

Omit LET because it is extraneous.

LIST from line number, to line number

When you want to LIST lines in a BASIC program in a memory, you use this command.

Immediate-mode example:

Type the following program, which you will use to perform the exercises below.

You type 10 REM EXAMPLE

You type 1000 ?"I"

You type 2000 ?"LOVE"

You type 3000 ?"ADAM"

To display the whole program,

You type LIST

To display line number 3000,

You type LIST 3000

To display line numbers 1000 through 3000,

You type LIST 1000,3000

To print the program on the printer,

You type PR#1:LIST:PR#0

Helpful hints:

Use the SPEED=150 command to slow the rate at which new lines appear on the screen to a readable speed.

You can only display 22 lines on the screen at one time.

LOAD program name,Ddrive

Use the LOAD command to copy a program into memory from tape or disk. The program will replace any previous program in memory.

The *program name* is required and must be a standard ADAM filename (1 to 10 characters long).

Ddrive. The D identifies the parameter as a drive. The left tape drive is 1, the right tape drive is 2, and the disk is 3. If the number is omitted, it defaults to the drive used last.

Immediate-mode example:

To LOAD the program PICMAKER from disk,
You type LOAD picmaker,D3

When the program is loaded, BASIC displays the] prompt, and you can RUN, change your program, or SAVE it on another drive or tape.

Helpful hints:

Use SAVE to copy a BASIC program to a disk or tape file.

Use CATALOG to display the name of the program on tape or disk.

Use RECOVER so the LOAD can access the backup version of the program.

LOCK filename, Ddrive

When you want to protect your file so the DELETE command will not delete the current version of the file from the tape or disk, and the SAVE will not replace the backup version, you use this command.

If you try to DELETE a locked file BASIC displays the message:

File Not Found

If you SAVE to a locked file, the SAVE will work but BASIC will mark the backup copy LOCKed. Subsequent SAVE will be stored in the file \$\$\$\$1. To UNLOCK the backup copy, temporarily RENAME the current version. RECOVER the backup copy to make it current.

Immediate-mode example:

```
You type      NEW
You type      10 REM VERSION 1
You type      SAVE version
Adam copies the program to tape
You type      CATALOG
ADAM displays Volume: FIRST DIR
               A 1 version
               252 Blocks Free
You type      LOCK version
You type      CATALOG
ADAM displays Volume: FIRST DIR
               *A 1 version
               252 Blocks Free
```


The asterisk (*) to the left of the A indicates that the file is LOCKed.

You type DELETE version

ADAM displays File Not Found

You can't DELETE a LOCKed file. You must UNLOCK it, to DELETE it.

Helpful hints:

Refer to RENAME to change the name of a locked file.

Refer to UNLOCK to unlock a LOCKed file.

LOG (arithmetic expression)

You use the LOG function to get the natural logarithm of the arithmetic expression. Logarithms provide a fast way of doing multiplication. Consider the series of powers of the number 2.

Power	8	7	6	5	4	3	2	1
Number	256	128	64	32	16	8	4	2

The powers or exponents are the logarithm of these numbers to the base 2.

To multiply any two numbers with different exponents but the same base, add the logarithm(exponents) together and then compute the value of the base raised to the logarithm sum.

For example:

Number	$8 * 16 = 128$
Base 2 equivalent	$2^3 * 2^4 = 2^7$
Logarithm	$3 + 4 = 7$

The Number row in the table shows the multiplication of 8 by 16. The Base 2 row shows the equivalent values represented as powers of two. The exponents 3 and 4 are the logarithm of the numbers to the base 2. The Logarithm row shows the exponents added together. Using the sum, 7, as the exponent on the base 2 gives the correct answer for 8 times 16: 128.

The LOG function converts a number to a logarithm using the natural number, identified mathematically as *e*, as the base. The EXP function converts the log back to a number.

Immediate-mode example:

To see how logarithms work using these BASIC functions, let ADAM do the arithmetic for the above illustration.

Type the following immediate-mode statements.

You type A=LOG(8)

You type PRINT A

ADAM displays 2.07944154

You type B=LOG(16)

You type PRINT B


```
ADAM displays 2.77258872
You type      PRINT EXP(A+B)
ADAM displays 128
ADAM got it right!
```

Helpful hints:

The conversion of number to log of the base e and exponentiation of the answer back to decimal takes time. For most multiplication requirements, it is not worth it. If your application has a lot of multiplication of numbers, you may find that adding logs requires less time than multiplication.

LOMEM: address

When you want to protect an area of memory for your own use, you use this command. At the end of the SmartBASIC Z-80 interpreter program, BASIC stores the data required by your program. With the LOMEM: command you tell SmartBASIC to store your program data at a higher address, making the memory space from the end of BASIC to the LOMEM: address available to you. This area is protected. The BASIC interpreter will not store any data or statements in this space. At this writing the SmartBASIC interpreter program ends below 28000. The *Companion* recommends that you store your own Z-80 programs and shape tables at this address. To protect the area, set LOMEM: to address equal to 28000 plus the length of your shape table or Z-80 program.

Immediate-mode example:

```
You type      ?FRE(0)
ADAM displays 25919
You type      LOMEM:29000
You type      ?FRE(0)
ADAM displays 24325
```

You reduced the workspace available to BASIC by the difference between the first FRE(0) and the second FRE(0).

Helpful hints:

See Chapter 9 on music and Appendix D on creating a shape table for examples of using LOMEM:.

Use the POKE command to put characters into the protected area and the PEEK command to get characters from the protected area.

The BSAVE command copies a protected area to a disk or tape file, and the BLOAD command copies a BSAVE file back into the protected area.

If you try to set LOMEM: to an area below the end of SmartBASIC, you will get the error message:

```
?OUT OF MEMORY
```


MID\$ (string, from, howmany)

You use the MID\$ function to extract a copy of the middle characters from a string variable or constant. The string can be a string constant, variable or expression. Use a whole number or arithmetic expression to define the *from* and *howmany* arguments. The *from* value defines from which position within the string, counting from the left, to begin extraction. The *howmany* value defines how many characters to extract; if omitted, all characters to the right end of the string are selected.

Immediate-mode example:

```
You type      AS="ABC456XYZ"
You type      ?MID$(AS,4,3)
ADAM displays 456
You type      ?MID$(AS,1,3)
ADAM displays ABC
You type      PRINT MID$(AS,7)
ADAM displays XYZ
```

Program example:

It is often easier to manipulate character strings when they are stored as an array of ASCII values, rather than as characters. This program uses the ASC and MID\$ function to convert each character in the string to an element in the *char%*.

```
10 REM mid
100 DIM char%(15)
1000 INPUT "STRING="; string$
2000 FOR position = 1 TO LEN(string$)
2100 char%(position) = ASC(MID$(string$, position, 1))
2200 NEXT position
3000 REM print array
3050 FOR position = 1 TO LEN(string$)
3100 PRINT "POS="; position;
3110 PRINT ",CHR="; CHR$(char%(position));
3120 PRINT ",ASCII VALUE="; char%(position)
3200 NEXT position
```

Helpful hints:

Refer to the other string functions LEFT\$, RIGHT\$, LEN, STR\$ and VAL.

The CHR\$ function converts a number to a character.

MON C,I,O,L

When testing, or debugging, a program that reads or writes to tape or disk file, use this MONitor command to display the tape and disk input/output file commands and/or the data BASIC reads or writes to the file.

C displays all commands like OPEN, READ, WRITE and CLOSE so you can check that they have the correct format.

I displays all data read from the file by the INPUT command.

O displays all data written to the disk by the PRINT command.

L displays each statement line as the LOAD command copies it into memory. For example, MON L:LOAD PICMAKER will display every line in the PICMAKER program as BASIC reads it into memory.

You can enter C, I, O and L control letters in any order, but commas must separate them. To turn off a display, use the control letter with the NOMON command.

Immediate-mode example:

To monitor file commands,

You type MON C

To stop monitoring file commands,

You type NOMON C

Program example:

While debugging a program you may want to leave a MON C,I,O command in your program.

```
10 REM mon
500 d$ = CHR$(4)
1000 PRINT d$; "mon c,i,o"
1100 PRINT d$; "open xxxxx"
1200 PRINT d$; "close xxxxx"
1300 PRINT d$; "nomon c,i,o"
1400 PRINT d$; "delete xxxxx"
```

Helpful hints:

You should always use the MONitor command to check out a program that reads or writes to a file.

NEW

Every time you want to type a new program, use this command in immediate mode to delete the current program and all values assigned to numeric and string variables.

Immediate-mode example:

Type these statements to see how the NEW command works.

You type 10 REM NEW EXAMPLE

You type LIST

ADAM displays 10 REM NEW EXAMPLE

You type NEW

You type LIST

ADAM displays nothing

The NEW command erased the single statement, line 10, from memory.

NORMAL

Use the NORMAL command when in the text mode, to revert back to the standard display of white characters on black background. The INVERSE command displays black characters on white background. The NORMAL command cancels the INVERSE display mode.

Immediate-mode example:

You type TEXT

You type INVERSE

ADAM displays the] prompt as a black character on white background.

You type NORMAL

ADAM displays the] as a white character on black background.

Program example:

Refer to the INVERSE program example.

Helpful hints:

You can use the INVERSE display character commands to attract attention or highlight important information. The NORMAL command returns character display to the standard mode.

NOT

You use NOT, called a logical operator, to reverse the truth of a value. If the value is true (one or nonzero), then NOT makes it false (zero). If the value is false (zero), then NOT makes it true (one).

Immediate-mode example:

The following examples illustrate how the NOT logical operator works.

You type ?NOT 0

ADAM displays 1

Zero represents false. The NOT operator reverses the truth of a value. The reverse of false is true, which BASIC represents by the value one.

You type ?NOT 5

ADAM displays 0

BASIC considers a nonzero value like 5 as having a true value. The NOT operator changes it from true to false. The false value is zero.

NOT has a higher precedence than AND and OR, which means that BASIC does NOT operations first.

You type ?NOT 0 OR 1

ADAM displays 1

The NOT first changes the 0 to 1, and then BASIC does 1 OR 1, which is true. By putting parentheses around the ORed values, you change the order in which BASIC computes the result.

You type ?NOT (0 OR 1)

ADAM displays 0

First BASIC calculates (0 or 1), which gives a true result(a value 1); then the NOT operator changes the true value to false (0).

Program example:

The `nor` program is a logical calculator, allowing you to enter a logical operator (the words NOT, OR and AND) and a value to compute the logical result of the entered value and the previous value in memory.

```

10 REM not logical calculator
500 result$ = "false": true% = 1
600 beep$ = CHR$(7)+CHR$(7)
1000 INPUT "not,or,and,end>"; logical$
1100 IF logical$ = "not" THEN 2000
1200 IF logical$ = "or" THEN 3000
1300 IF logical$ = "and" THEN 4000
1350 IF logical$ = "end" THEN END
1400 PRINT beep$; "only not,or,and valid"
1500 GOTO 1000
2000 REM not
2200 result% = NOT result%
2250 condition$ = result$: result$ = ""
2300 GOSUB 6000: REM print result
2900 GOTO 1000
3000 REM or
3050 GOSUB 5000: REM get condition
3100 result% = result% OR condition%
3200 GOSUB 6000: REM display result
3900 GOTO 1000
4000 REM and
4100 GOSUB 5000: REM get condition
4300 result% = result% AND condition%
4400 GOSUB 6000: REM display result
4900 GOTO 1000
5000 REM get condition
5100 INPUT "true or false>"; condition$
5200 IF condition$ = "true" THEN condition% = true%: RETURN
5300 IF condition$ = "false" THEN condition% = false%: RETURN
5400 GOTO 5100
6000 REM display result
6010 PRINT result$; " ";
6050 result$ = "false"
6100 IF result% THEN result$ = "true"
6200 PRINT logical$; " "; condition$; "="; result$
6900 RETURN

```


Helpful hints:

Refer to the other logical operators OR and AND.

Where possible make positive tests because you understand them more quickly than using NOT to reverse the truth of a negative.

The following produce statements giving identical results, but the first is easier to understand. Notice that removing the parentheses changed the logical operator from OR to AND to produce the same logical test.

You type ?NOT (1 OR 1)

ADAM displays 0

You type ?NOT 1 and NOT 1

ADAM displays 0

NOTRACE

Use this command to turn off the TRACE mode. See TRACE.

ON selection value GOSUB line number, line number, . . .

This command allows you to select the subroutine you want executed by number. It acts like a multiway switch with the selection value as the switch pointing to the line number BASIC will execute next. The selection value can be a numeric variable or an arithmetic expression. If the selection value has an integer value of one, BASIC transfers execution to the first line number, the second if the value is two, etc. The subroutine must exit with a RETURN to transfer back to the statement following the ON . . . GOSUB. A zero value of the arithmetic expression or a number greater than the line-number list count causes program execution to proceed to the next statement.

Program example:

Many times you want the program to do several functions, controlled by pressing a single key—maybe one of the six function keys. This program demonstrates how to use the value of the function key to execute the desired subroutine. It uses the GET command to read a function key value. If you press either function key I or II the program displays the name of the key pressed.

```

10 REM  ongosub
1000 PRINT "Press function key I or II"
1100 GET key$: key% = ASC(key$)
1200 IF key% = 3 THEN END
1300 IF key% = 129 OR key% = 130 THEN 1500
1400 PRINT "YOU PRESSED THE WRONG KEY!": GOTO 1000
1500 ON key%-128 GOSUB 2000, 3000
1900 GOTO 1000
2000 REM  function I
2100 PRINT "FUNCTION I"
2900 RETURN
3000 REM  function II
3100 PRINT "FUNCTION II"
3900 RETURN

```


Helpful hints:

See Chapter 7 for a practical application of this technique.

See POP to see how to exit from a subroutine without doing a RETURN.

The *Companion* recommends that you use the ON . . . GOSUB instead of the ON . . . GOTO because the program logic flow is easier to follow if you know the subroutine will always return to the statement after the ON . . . GOSUB. A subroutine entered by an ON . . . GOTO can terminate by going to any other statement in the program.

ON selection value GOTO line number, line number, . . .

This command allows you to select the routine you want to execute by number. It acts like a multiway switch with the selection value as the switch pointing to the line number BASIC will execute next. The selection value can be a numeric variable or an arithmetic expression. A selection value of one selects the first line number on the left, two selects the second line number, etc. BASIC does a GOTO to the selected line number. A zero value of the arithmetic expression or a number greater than the line-number list count causes program execution to proceed to the next statement.

Program example:

Many times you want the program to do several functions depending on the function key pressed. This program uses the GET command to read the value of the key pressed. The program supports two functions: *move* and *insert*. If you press either key, the program displays a message identifying the key pressed.

```

10 REM   ongoto
1000 PRINT "Press insert or move key"
1100 GET key$: key% = ASC(key$)
1200 IF key% = 3 THEN END
1300 IF key% = 148 THEN key% = 1: GOTO 1500
1350 IF key% = 146 THEN key% = 2: GOTO 1500
1400 PRINT "YOU PRESSED THE WRONG KEY!": GOTO 1000
1500 ON key% GOTO 2000, 3000
1900 STOP: REM   should never gethere
2000 REM   insert
2100 PRINT "INSERT"
2900 GOTO 1000
3000 REM   move/copy
3100 PRINT "MOVE/COPY"
3900 GOTO 1000

```

Helpful hints:

Only use this command as a multiway switch. The *Companion* recommends that you use the ON . . . GOSUB as the preferred technique for conditional execution of a subroutine.

ONERR GOTO line number

Use the ONERR statement to avoid an error message that halts the execution of your program when an error does occur. You can write your program to recover from many types of errors.

Program example:

This program demonstrates how to handle the FILE NOT FOUND error that occurs when the operator misspelled the filename the program must process.

```

10 REM onerr goto
900 true% = 1: d$ = CHR$(4)
1000 INPUT "FILENAME>"; file$
1050 IF LEN(file$) = 0 THEN END
1100 GOSUB 9000: REM open file
1200 IF found% THEN 1500
1300 PRINT "FILE "; file$; " NOT FOUND"
1400 GOTO 1000
1500 PRINT "FILE "; file$; " FOUND"
1600 GOTO 1000
9000 REM open file
9005 ONERR GOTO 9100
9010 PRINT d$; "open "; file$
9020 PRINT d$; "read "; file$
9030 INPUT "; record$
9040 CLRERR: found% = true%
9050 PRINT d$
9090 RETURN
9100 REM onerr routine
9110 CLRERR: IF ERRNUM(0) = 5 THEN 9200
9120 PRINT "ERRNUM="; ERRNUM(0)
9190 RETURN
9200 REM file not found
9210 PRINT d$; "close "; file$
9220 PRINT d$; "delete "; file$
9230 found% = false%
9290 RETURN

```

Helpful hints:

CLRERR commands SmartBASIC to handle errors, rather than GOTO the ONERR routine.

RESUME command returns execution to the line in which the error originally occurred.

ERRNUM (0) returns the error number.

You should use an ONERR routine to handle operator or file-caused errors. You should handle the following errors in your program.

Error Number	Description
5	END OF DATA—You've reached the end of data in file.
6	FILE NOT FOUND—ADAM can't find the file by the name used in the OPEN statement.

- 254 BAD RESPONSE TO INPUT—If the program requested integer or decimal INPUT and the operator entered bad data
- 255 BREAK—Caused by the operator pressing CTL-C and you want to execute a special termination routine or not allow termination by CTL-C.

OPEN filename, Llength, Ddrive

To identify the file you want to READ or WRITE, you use this command. For compatibility with Applesoft BASIC, SmartBASIC requires that you define the OPEN command as an item in a PRINT command list item with the value defined by CHR\$(4) as the first item in the list.

Example:

Assume that DS=CHR\$(4)

```
1000 PRINT DS;"ADAM,L80,D1"
```

The *filename*, here ADAM, is required and must be a standard ADAM filename of 1 to 10 characters with no spaces.

Llength. The L identifies the parameter as the length in characters of each record in a random file. It includes the RETURN character BASIC places at the end of PRINT line written to tape or disk. When you create a random file, BASIC sets aside space on the tape or disk equal to the length. To READ a random file, BASIC uses the record number and length to calculate the location of the record in the file. For sequential files, don't use the L parameter.

Ddrive. The D identifies the parameter as a drive number. The left tape drive is 1, the right tape drive is 2, and the disk drive is 3. If the number is omitted, BASIC defaults to the drive used last.

Program example:

This program creates a three-record random file. Use the CATALOG command to verify that the program created the file.

```
10 REM open
500 d$ = CHR$(4)
600 DATA BENSON,ROCHESTER,DOUGLAS
1000 PRINT d$; "open companion,L50"
1100 FOR record = 0 TO 2
1200 PRINT d$; "write companion,R"; record
1300 READ record$
1400 PRINT record$
1500 NEXT record
1600 PRINT d$; "close companion"
```


Helpful hints:

Use APPEND instead of the OPEN command to add records to the end of a file.

Always remember to CLOSE a file with the same name you OPENed it with. CLOSE sets the directory size to the actual file size; otherwise when you're creating a file it will have the largest possible size.

Refer to READ, WRITE, INPUT, and PRINT for examples of file processing.

OR

You use OR, called a logical operator, to test if either of two values is true or false. A true result returns a value one, and a false result returns a value zero. The test, called a compound logical expression, compares either numeric variables (for example, *done%*), arithmetic expressions (for example, *counter% - 4*) or relational test (for example, *counter% = 56*). The following truth table summarizes the possible combinations and results.

<i>Value 1</i>	OR	<i>Value 2</i>	<i>Result</i>
true		true	true
false		true	true
true		false	true
false		false	false

Immediate-mode example:

Try the following examples of compound logical expressions formed with the OR operator.

You type ?1 OR 0

ADAM displays 1

The value one means true. True or false gives a true result.

You type false%=0

You type PRINT false% OR 6<3

ADAM displays 0

The value zero means false. A false (*false%=0*) OR false (it's false that 6 is less than 3) gives a false result.

You type ?2 OR 3

ADAM displays 1

A true result because any nonzero value is true.

Program example:

The following program loops until you press either button on hand controller 1. Reading a PDL function value resets its value to zero.

```

10 REM      or
1000 REM    loop
1100 left% = PDL(7)
1200 right% = PDL(9)
1250 PRINT "LEFT="; left%; " RIGHT="; right%
1500 IF left% OR right% THEN END
1600 GOTO 1000

```


Helpful hints:

Normally you use the OR logical operator in an IF . . . THEN statement to determine the combined truth of two values.

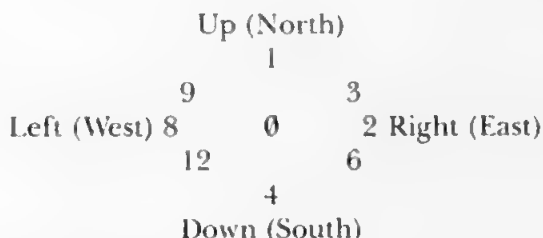
Refer to the other logical operators AND and NOT.

PDL (function number)

The two ADAM hand controllers operate as keypads, buttons, paddles or joysticks. The function number inside the parentheses determines what information you want BASIC to tell you about the controller.

Use paddle functions 0 to 3 when you want to use the joysticks as paddles. Paddles on other computers are wheels or knobs that turn right and left. When you turn the wheel to the extreme left, the PDL function returns a value zero. As you turn the wheel to the right, the value returned becomes greater until at the extreme right the PDL function returns a value of 255. With ADAM, each controller acts as two paddle wheels. Pressing the controller left and right simulates the turning of one paddle wheel, while pushing up and down simulates the left/right turning of the other paddle wheel. Use PDL(1) and PDL(3) to return the values of the left/right and up/down paddles on hand controller 1 (the closest one to the front of the memory console). Use PDL(0) and PDL(2) to return the values of the left/right and up/down paddles on hand controller 2.

With joysticks, you're interested in which direction the player has pushed the stick, not the position of the wheel. As you game players know, you can push the ADAM controller in any one of the eight directions shown in the picture below.



0 = Straight (no push in any direction)

1 = Pushed up (north)

3 = Pushed up and to the right (northeast)

2 = Pushed right (east)

6 = Pushed down and to the right (southeast)

4 = Pushed down (south)

12 = Pushed down and to the left (southwest)

8 = Pushed left (west)

9 = Pushed up and to the left (northwest)

Use PDL(5) to return the direction value for controller 1 and PDL(4) to return the direction value for controller 2.

The table below summarizes the 15 PDL functions.

<i>Function</i>	<i>Controller</i>	<i>Type data</i>	<i>Value returned</i>
PDL(0)	2	Paddle wheel	Moving joystick up returns
PDL(1)	1		lower values until 0. Moving joystick down returns higher values until 255.
PDL(2)	2	Paddle wheel	Moving joystick left
PDL(3)	1		returns lower values until 0. Moving joystick right returns higher values until 255.
PDL(4)	2	Joystick	Indicates direction
PDL(5)	1		joystick moved by player 0 = Straight up (not pushed) 1 = Pushed up 2 = Pushed right 3 = Pushed up to right 4 = Pushed down 6 = Pushed down to right 8 = Pushed left 9 = Pushed up to left 12 = Pushed down to left
PDL (6)	2	Left button	0 = not pressed
PDL (7)	1		1 = pressed
PDL (8)	2	Right button	0 = not pressed
PDL (9)	1		1 = pressed
PDL (10)	2	Keypad	ASCII value of number or
PDL (11)	1		symbol pressed. Keypad number 0-9 returns value 48-57. # returns value 35; * returns value 42. No key pressed returns value 0.
PDL (12)	2	Keypad	Binary value of number
PDL (13)	1		Keypad number 0-9 returns 0-9. # returns value 11; * returns value 10. No key pressed returns value 15
PDL (14)	2	Reserved for Rollerball	Not currently implemented
PDL (15)	1		

Immediate-mode example:

PDL functions 4 through 13 return a nonzero value only when the button, keypad square or joystick is pressed. For example, press down on the * key on controller 1 and hold it down while pressing the RETURN key.

You type ?PDL(11) (hold down the * key)

ADAM displays 42

The ASCII value for asterisk (*) is 42. Use the CHR\$(42) function to convert the number to a character.

You type ?CHR\$(42)

ADAM displays *

The value 42 represents the asterisk symbol.

Program example:

For any nonzero controller value returned, the following program prints a line identifying the controller, the function number and the value returned. RUN it to learn how the controller returns values to a BASIC program.

```

10 REM   pdl
100 DIM array%(14)
500 true% = 1
1000 FOR hand = 0 TO 1
1100 FOR func = hand TO 10+hand STEP 2
1200 value% = PDL(func)
1250 IF func < 4 AND array%(func) = value% THEN 1400
1300 IF value% THEN GOSUB 2000
1400 NEXT func
1510 value% = PDL(12+hand)
1520 IF value% <> 15 THEN GOSUB 2000
1530 IF value% = 11 THEN finished% = true%
1700 NEXT hand
1900 IF NOT finished% THEN 1000
1950 END
2000 REM   print controller value
2100 PRINT "hand="; hand; ",func="; func; ",VALUE="; value%
2200 array%(func) = value%
2900 RETURN

```

Helpful hints:

Unlike the INPUT and GET commands, the PDL function does not wait for input; it just returns whatever value it finds. For this reason in certain applications, and not exclusively games, you may want to use the joystick instead of the keyboard arrow keys to move the cursor or some other video figure around the screen.

Because the PDL function returns a nonzero only when an action takes place, like pressing the button, if the processing program does not execute the PDL function while the action takes place, the program never recognizes that the event occurred. This is why you sometimes press a game button and nothing happens.

PEEK (decimal address)

Use the PEEK function to look at any memory location from 0 to 65535. The value returned is a numeric value from 0 to 255. You can use a negative integer variable to reference a memory location greater than 32767, the maximum positive integer. To compute the negative number, subtract 65536 from the positive address greater than 32767. For example, to store the memory address 51001 as a negative number in the integer variable *address%*,

You type *address%=51001-65536*

You type *?address%*

ADAM displays *-14535*

The numbers 51001 and - 14535 address the same memory location.

Immediate-mode example:

Memory locations 16766 and 16767 store the address of the shape table used by the DRAW and XDRAW commands. By PEEKing at these locations, you can determine the address of the first character in the Coleco-supplied shape table. The Z-80 microprocessor stores all addresses in reverse order with the low-order byte at the lower address—in this case 16766. To get the value of the address, you must multiply the high-order byte by 256 and add it to the low-order byte.

You type *?PEEK(16766)*

ADAM displays *206*

This is the low-order byte of the address. Note: Your version of BASIC may show a different number because it has its shape table located at a different address.

You type *?PEEK(16767)*

ADAM displays *103*

This is the high-order byte. Now compute the shape-table address.

You type *?256*PEEK(16767)+PEEK(16766)*

ADAM displays *26574 (shape-table address)*

One byte represents 256 values and has a positional value 1, but two bytes can represent 65536 values, and the second byte has a positional value of 256. That's why you multiply the high-order byte by 256.

The FOR . . . NEXT loop allows you to PEEK at the bytes in the built-in shape table.

You type *ST=26574 (shape-table address)*

You type *FOR a=st TO ST+13:?PEEK(a):next a*

ADAM displays *1 (number of shapes—only 1)*

0

4 (relative location of first shape)

0

54 (first shape)

63

36


```

36
45
45
54
54
63
0   (end of shape indicator)

```

Refer to Appendix D for more information on shape tables.

Program example:

This program retrieves the data stored by the POKE program (see the program example for the POKE command) in the protected memory below low memory (28000 to 28999). The first byte, at 28000, defines the number of the characters that follow it. The PEEK takes the character at the address and puts it into an integer which the CHR\$ function converts back into a character.

Type both the PEEK and POKE programs and store them on a tape. RUN the POKE program, which allows you to enter the data to pass to the PEEK program. After storing the data, the POKE program automatically RUNs the PEEK program.

```

10 REM peek
20 LOMEM :29000
100 d$ = CHR$(4)
1000 address% = 28000
1100 PRINT "PEEK"
1150 size% = PEEK(address%)
1160 address% = address%+1
1200 FOR p = 1 TO size%
1210 message$ = message$+CHR$(PEEK(address%))
1230 address% = address%+1
1290 NEXT p
1300 PRINT message$
1310 PRINT "press any key to continue"
1320 GET key$
1390 PRINT d$; "run poke"

```

Helpful hints:

You use PEEK to retrieve data you previously stored in memory with POKE.

PLOT column, row

In the low-resolution graphics mode (GR), you use this command to place a colored square on the video screen at a particular column and row. The

last-executed COLOR command determines the color square displayed. If the program did not execute a COLOR command, you won't see the square because the default color is black. Also, you won't see the effect for this command unless the program has executed the GR command to display the low-resolution graphics memory instead of the standard TEXT screen.

The low-resolution graphics screen consists of a grid of 40 columns by 40 rows. Columns are numbered from left to right from 0 to 39. Rows are numbered top to bottom from 0 to 39. If either the column or row value is negative or greater than 39, SmartBASIC displays the error message:

?ILLEGAL QUANTITY

In this mode, BASIC uses the bottom four lines of the screen to display text from PRINT commands.

Immediate-mode example:

```
You type      GR
You type      COLOR=11 (light red or pink)
You type      COL=20
You type      FOR ROW=0 TO 39 STEP 2:PLOT COL,ROW:NEXT
               ROW
```

ADAM displays a column of pink squares in the middle of the screen.

Program example:

This program displays a big letter A on the screen.

```
10 REM plot letter 'A'
100 DATA 0,0,1,0,0
110 DATA 0,1,0,1,0
120 DATA 1,0,0,0,1
130 DATA 1,0,0,0,1
140 DATA 1,1,1,1,1
150 DATA 1,0,0,0,1
160 DATA 1,0,0,0,1
900 GR
950 COLOR = 11
1000 FOR row = 11 TO 17
1100 FOR column = 21 TO 25
1200 READ dot%
1300 IF dot% THEN PLOT column, row
1400 NEXT column
1500 NEXT row
```

Helpful hints:

Chapter 7 explains how to write a picture-maker program using low-resolution commands. This program allows you to draw a color picture on the screen and save and retrieve it from the data cassette.

POKE memory address, value

Use the POKE command to store a value in a memory location. The address of the memory location can range from 0 to 65535. Because the high-order bit of an integer acts as a sign, a negative number less than 32767 has an equivalent value greater than 32767. So -16384 is equivalent to 49152. The value stored in memory can range from 0 to 255. Use a number or arithmetic expression to define memory address and the value.

Immediate-mode example:

```
You type      HIMEM:52000
You type      POKE 52001,ASC("A")
You type      PRINT CHR$(PEEK(52001))
ADAM displays A
```

Program example:

This program allows you to enter characters from the keyboard and store them in memory beginning at location 28001. The number of characters POKEd into memory is stored at address 28000. Finally, the POKE program RUNs the PEEK program (see the PEEK command program example), which retrieves and displays the data. The POKE and PEEK programs demonstrate the technique of passing data from one program to another.

```
10 REM      poke
20 LOMEM :29000
100 d$ = CHR$(4)
1000 HOME: PRINT "POKE"
1050 address% = 28000
1100 INPUT "enter message>"; message$
1150 POKE address%, LEN(message$)
1160 address% = address%+1
1200 FOR p = 1 TO LEN(message$)
1210 integer% = ASC(MID$(message$, p, 1))
1220 POKE address%, integer%
1230 address% = address%+1
1290 NEXT p
1300 PRINT d$; "run peek"
```

Helpful hints:

You can use POKE to store shape tables and machine-language instructions defined in DATA statements. You can pass data from one program to another by using POKE to store each character of a string in the area protected from BASIC. Once loaded BASIC doesn't reset LOMEM:, so the next program can retrieve the data using the PEEK commands. Refer to the RUN command for an example.

See the CALL command for an example of how to use POKE to place a

Z-80 machine-language program into memory.

Refer to Appendix D for an example of how to POKE a shape table into memory.

POP

The POP function removes the return line number from BASIC's internal stack, effectively changing the most recent GOSUB into a GOTO after the fact.

Program example:

You use POP when you're deep in a low-level subroutine and want to get to a higher level without going through the levels in between. Usually you do this when an error occurs that aborts all other processing but you don't want to end the program. The following trivial program demonstrates this situation. You should find it similar to the pocket calculator in Chapter 5. Notice what happens if you delete statement 8100 and you don't enter any AMOUNT.

```

10 REM pop example
1000 GOSUB 2000: REM get input
1990 GOTO 1000
2000 INPUT "AMOUNT="; amount$
2100 GOSUB 2500: REM edit amount
2200 total = total+VAL(amount$)
2300 PRINT "total="; total
2490 RETURN
2500 IF LEN(amount$) = 0 THEN 8000
2600 RETURN
8000 REM clear up stack
8100 POP: POP
8200 GOTO 10: REM try again

```

Helpful hints:

Don't execute the RETURN statement if you executed POP.

Avoid using POP because it makes it difficult for someone reading the program to understand what's happening. If you do use it, explain what happens with a REM statement.

POS(0)

Use this command, in the TEXT display mode, to get the location of the horizontal cursor. The numeric value returned has a range 0 to 30 and is one less than the actual column used by the HTAB command.

Immediate-mode example:

You type HTAB 13: ?POS(0)
 ADAM displays 12

The HTAB command positions the cursor to text column 13, so POS(0) returns a value 1 less than 13, or 12.

Helpful hints:

You won't have much use for this command.

VPOS returns the row or vertical cursor location.

POSITION filename,Rskipnumber

In a sequential file, when you want to skip forward over fields rather than read them with an INPUT command, you use this command. Since POSITION cancels the effect of a previously executed READ command, you must follow the POSITION command with a READ command. Like the other file commands, use the PRINT command to pass the POSITION command to BASIC. For example,

```
1000 PRINT DS;"POSITION magazines,R2"
```

where the variable DS=CHR\$(4).

The filename, *magazines* in the example, is required and must be a standard ADAM filename consisting of 1 to 10 characters with no embedded spaces.

Rskipnumber. The R identifies the parameter as a skip number. When you use the PRINT command to write a list of variables to a file, each variable is considered a field. Suppose you wrote out three variables or fields, and now you want to INPUT only the third field. You specify the *skipnumber* as two to skip over the first two fields and INPUT the third.

Program example:

This program writes a sequential file named *magazine* consisting of nine records. For each magazine, the program writes three variables as records: *month\$*, *year\$*, and *name\$*. After closing the file, it OPENS the file, which tells BASIC to start READING from the beginning of the file. Then it uses the POSITION command to skip over the *month\$* and *year\$* records and INPUT only the *name\$* record. Notice that the program stores the *name\$* in a string array until it reads all the records. Then it uses a FOR . . . NEXT loop to PRINT the names on the screen. With sequential files you can't INPUT and then PRINT to the screen and printer until the program has CLOSEd the file.

```
10 REM position
500 d$ = CHR$(4)
2000 REM create magazine
2100 PRINT d$; "open magazine"
2200 PRINT d$; "write magazine"
2300 FOR record = 1 TO 3
```



```

2400 READ month$, year, name$
2500 PRINT month$
2510 PRINT year
2520 PRINT name$
2600 NEXT record
2800 PRINT d$; "close magazine"
3000 REM magazine
3100 PRINT d$; "open magazine"
3200 PRINT d$; "read magazine"
3250 FOR record = 1 TO 3
3300 PRINT d$; "position magazine,r2"
3400 INPUT ""; name$(record)
3500 NEXT record
3800 PRINT d$; "close magazine"
5000 REM data statements
5100 DATA Jan,1983,BYTE
5200 DATA Jan,1983,NEWSWEEK
5300 DATA Mar,1983,TIME

```

Helpful hints:

The *Companion* recommends that you use random-access techniques to READ your file because you can display on the screen and/or printer without having to wait until the file is closed. Refer to Chapter 10, which explains a mailing-label system illustrating this technique.

Refer to the other file commands OPEN, APPEND, CLOSE, READ, WRITE, PRINT and INPUT.

PRINT data list (or ?data list)

The question mark (?) is a shorthand way of writing the PRINT command.

The PRINT command displays the data list on the video screen or printer, or writes it to the tape or disk. The data list may contain the following items:

<i>Item</i>	<i>Example</i>
string constants	"TOTAL="
numeric constants	123.45
string variables	amount\$
integer variables	amount%
decimal variables	amount
elements in arrays	amount(1)
string expressions	TOTAL+LEFT\$(amount\$,4)
arithmetic expressions	RND(-amount)*5
logical expressions	true% OR false%
relational expressions	5>7
SPC function	SPC(4) print 4 spaces
TAB function	TAB(14) column 14

BASIC computes the value of expressions before displaying them, and because the PRINT command displays all values as character strings, where necessary it converts numeric values to strings.

A comma (,) or semicolon (;) separates items on the data list. Refer to the immediate-mode examples for what they do.

The PRINT command has several uses.

1. In the immediate mode use the PRINT statement to display the contents of variables on the video screen.

You type A\$="ABC"

You type PRINT A\$

ADAM displays ABC

2. In the immediate mode use the PRINT statement with arithmetic expressions to do simple calculations.

You type PRINT 256*256

ADAM displays 65536

3. In a program use PRINT to display text on the video screen or the daisy-wheel printer.

NEW

10 REM PRINT TO DAISY WHEEL PRINTER

1000 PR#1: REM OUTPUT TO PRINTER

1100 PRINT "POWER VALUE"

1200 FOR POWER=1 TO 16

1300 PRINT POWER;TAB(5);2^POWER

1400 NEXT POWER

1500 PR#0: REM RETURN OUTPUT SCREEN

RUN

4. In a program use the PRINT statement with CHR\$(4) to pass input/output commands to BASIC.

10 REM PRINT2 CREATE A FILE

100 DS=CHR\$(4)

900 INPUT "FILENAME=":name\$

950 IF LEN(NAMES)=0 THEN END

1000 PRINT DS;"OPEN ":name\$

2000 PRINT DS;"WRITE ":name\$

3000 PRINT "HELLO"

4000 PRINT DS;"CLOSE ":name\$

Immediate-mode examples:

A comma (,) between print fields causes a skip to the next print zone.

You type PRINT "1234","ABCD"

ADAM displays 1234 ABCD

A semicolon between print fields causes no skip.

You type PRINT "1234";"ABCD"

ADAM displays 1234ABCD

Program example:

A semicolon after the last print field prevents a return to the beginning of

the next line. This allows more than one print statement to print on the same physical line. Try the following program.

```
NEW
10 REM PRINT TWO PRINT STATEMENTS PUT DATA ON ONE
LINE
1000 PRINT -1;TAB(5);"ABC";SPC(2);
2000 PRINT "ON SAME LINE"
RUN
```

ADAM displays -1 ABC ON SAME LINE

Helpful hints:

Try all the examples. You will use the PRINT statement a lot, so you must understand what it can do for you.

Refer to the PRINT functions TAB and SPC, which you use only within a PRINT statement.

Refer to TEXT, NORMAL and INVERSE.

PR#number

The PR#1 command directs PRINT command lines to the printer. The PR#0 command stops PRINT command lines from going to the printer.

Immediate-mode example:

To LIST a program on the printer,

You type PR#1:LIST:PR#0

To print the tape catalog on the printer,

You type PR#1

You type CATALOG

You type PR#0

Program example:

The PR program prints two lines on the printer. The PRINT CHR\$(4) statement stops display lines from going to the printer. If you use PR#0, the command would print before stopping printed output.

```
10 REM pr#1
1000 PR #1
1050 PRINT
1100 ten$ = "1234567890"
1200 FOR `c = 1 TO 8
1300 PRINT ten$;
1400 NEXT c
1450 GOTO 1200
1500 PR #0
```

Helpful hints:

CTRL-S makes the printer pause; CTRL-Q starts it again.

READ variable list

When you want to print, or assign to variables the values defined in a DATA statement, you use the READ command. A comma (,) separates each item in the variable list. The variable or array element name must have the same data type as the DATA item. For example, you can't READ a string DATA item like ABCD into a decimal variable like *amount*. SmartBASIC will give you an error message:

```
?ILLEGAL QUANTITY
```

Helpful hints:

See DATA and RESTORE.

READ filename,Rrecordnumber

This is a file command given from within a PRINT statement. With this command you tell BASIC to get ready to READ a record from tape or disk. The next INPUT command lists the variables actually read. For example,

```
1000 PRINT DS;"READ chasewords,R0"
```

```
1010 INPUT records
```

The *filename*, here *chasewords*, is required and must be a standard ADAM filename consisting of from 1 to 10 characters with no embedded spaces.

Rrecordnumber. Required to read a file randomly. To read a file sequentially, omit this parameter.

The INPUT statement reads the data from the tape or disk into the variable *records*.

Sequential-file example:

When you READ a sequential file, records are read from the file in the order they are stored on the file.

To read a sequential file you need to use four commands:

OPEN—Defines the filename and drive.

READ—defines the operation as opposed to WRITE. It tells BASIC that all data for the INPUT command will come from the tape or disk file.

INPUT—Reads the data into the variables in its list. You must use a null prompt, two double quotes (" "), to prevent BASIC from displaying the question mark (?) on the screen each time BASIC executes the INPUT statement. For example,

```
INPUT "";records
```

The following program reads the sequential file created by the WRITE command example.


```

10 REM      readseq
100 DIM record$(100)
500 d$ = CHR$(4)
600 PRINT d$; "mon c,i"
1000 PRINT d$; "open sequential"
1100 PRINT d$; "read sequential"
1200 FOR record = 1 TO 3
1400 INPUT ""; record$(record)
1500 NEXT record
1600 PRINT d$; "close sequential"
1700 PRINT d$; "nomon c,i"
2000 FOR record = 1 TO 3
2100 PRINT record$(record)
2200 NEXT record

```

Random-file example:

When you READ a record randomly, you read by record number. You can read records in any order, even backward if you supply the record numbers in reverse order. For example, to read a three-record file backward, use record numbers 3, 2 and 1.

To read randomly, you must use two commands as a pair—for example, a PRINT command beginning with a DS variable [DS=CHR\$(4)] followed by the READ command character string to define the operation and record number. Use the INPUT command, the other half of the pair, to define the variables that store the data read.

A PRINT command with only a DS variable tells BASIC to read the next data from the keyboard, not the tape or disk file. This allows you to enter data to change the information read before using the WRITE command to place the data back on the tape at the same location. Chapter 10, which describes a mailing-label system, and the example below demonstrate this procedure.

This program reads the same three records as the WRITERAND program (see page 377), but reads them randomly in reverse order.

```

10 REM      readrnd
500 d$ = CHR$(4)
600 PRINT d$; "mon c,i"
1000 PRINT d$; "open random,L30,D1"
1200 FOR record = 3 TO 1 STEP -1
1350 PRINT d$; "read random,R"; record
1400 INPUT ""; record$
1420 PRINT d$: REM terminate command mode
1450 PRINT "RECORD>"; record$
1500 NEXT record
1600 PRINT d$; "close random"
1700 PRINT d$; "nomon c,o"

```

Helpful hints:

Refer to other file-handling commands OPEN, APPEND, WRITE and CLOSE for more information about processing files.

RECOVER filename,Ddrive

The second time you SAVE a program, SmartBASIC marks the first version as a backup copy. When you want to make the backup copy of the current version, use the RECOVER command.

The *filename* is required and must be a standard ADAM filename of 1 to 10 characters with no embedded spaces.

Ddrive. The D identifies the parameter as a drive number. The left tape drive is 1, the right tape drive is 2, and the disk is 3. If the number is omitted, BASIC defaults to the drive used last.

Immediate-mode example:

To RECOVER a backup copy, you must DELETE or RENAME the current version. To test for yourself how the RECOVER command works do the following.

```
You type      10 REM hello
You type      1000 ?"HELLO"
You type      SAVE hello,D1 (1 is the left tape drive)
```

Now replace the print statement to display GOODBYE.

```
You type      1000 ?"GOODBYE"
You type      SAVE hello
You type      CATALOG
ADAM displays VOLUME: FIRST DIR
              a 1 hello
              A 1 hello
```

The letter in the VOLUME column identifies the file. If it is uppercase it indicates the current file; if it is lowercase it indicates the backup copy. The number preceding the filename indicates how many of the 253 available blocks are used in storing the file on the tape.

To RUN or LOAD the backup copy you must first RENAME or DELETE the current version of *hello* and issue the RECOVER command.

```
You type      RENAME hello,goodbye
```

You changed the name of the current version of *hello* to *goodbye*. Now no current version of *hello* exists, so you can issue the RECOVER command.

```
You type      RECOVER hello
You type      CATALOG
ADAM displays VOLUME: FIRST DIR
              A 1 hello
              A 1 goodbye
```

The RECOVER command changes the lowercase a to a capital A. The capital A means the file is the current version. You can now RUN the first version of *hello*.

```
You type      RUN hello
ADAM displays HELLO
```


Helpful hints:

Don't rely on this command to RECOVER a program. Always SAVE a second copy of the program on another tape or disk.

REM (comment)

Use the REM command to put a comment in the program. Comments explain what the program does and why. They are helpful when you read the program to understand what it does.

Program example:

This program illustrates several places you can place REM statements in a program. You should place one at the beginning, another on the same line as a GOSUB, and one at the beginning of a subroutine.

```
10 REM rem to idenify program
1000 GOSUB 2000: REM name of subroutine
1990 END
2000 REM name of subroutine
2900 RETURN
```

Helpful hints:

The *Companion* strongly recommends that you use comments in your program. Although they do take up space, you should remove them only when you reach the maximum space available for a BASIC program.

RENAME from filename, to filename

To change a filename in the tape or disk directory, you use this command. The contents of the file remain unchanged.

Immediate-mode example:

You can't use a drive parameter with the RENAME command, so execute a command like SAVE, LIST or CATALOG with the Ddrive parameter to select the unit. To illustrate the RENAME command, create a single statement file called *big* and change its name to *small*.

```
You type      10 REM big
You type      SAVE big,D1 (1 is the left tape drive)
You type      RENAME big,small
You type      LOAD small
You type      LIST
ADAM displays 10 REM big
```

The RENAME command changed the name of the file, but the REM identification name still has the word *big*. When you change a program filename, also remember to change the name in the REM program identification statement.

Helpful hints:

Refer to the other commands like DELETE, RECOVER, SAVE, LIST and CATALOG that use the directory.

RESTORE

Use RESTORE when you want to reread the DATA statements from the beginning. As the READ command processes each item in the DATA statement list, BASIC updates a READ pointer to point to the next DATA item. RESTORE tells BASIC to reset the READ pointer to the beginning of the DATA list, the first item in the first DATA statement in the program.

Program example:

To make it easier for the operator, the program allows you to INPUT a code or name instead of the number. DATA statements define the conversion table. This program, similar to the routine in PICMAKER, allows you to enter the name of a high-resolution color and have the program draw a colored vertical bar. Instead of storing the names in an array, the program uses the RESTORE command to position the READ pointer to the beginning of the DATA list. In this case, RESTORE positions the pointer to the first data item in the DATA statement at line 1000.

```

10 REM ^ hplot animation sample
1000 HGR: orange = 5: row% = 100
1100 FOR column = 24 TO 224 STEP 16
1125 column% = column
1150 HCOLOR = orange
1200 GOSUB 3000: REM hplot face
1250 FOR delay = 1 TO 300: NEXT delay
1300 HCOLOR = black%
1400 GOSUB 3000
1500 NEXT column
1800 TEXT
1900 END
3000 REM HPlot face
3110 HPLOT column%, row% TO column%+15, row% TO column%+15, row%+15 TO column%,
row%+15 TO column%, row%
3120 HPLOT column%+6, row%+4: HPLOT column%+9, row%+4
3130 HPLOT column%+8, row%+7 TO column%+8, row%+9
3140 HPLOT column%+5, row%+10: HPLOT column%+11, row%+10
3150 HPLOT column%+5, row%+11 TO column%+11, row%+11
3190 RETURN

```

Helpful hints:

If the program contains DATA statements, you should place a RESTORE command at the beginning of your program-initialization routine so if the operator or player decides to play the game again at the end of the program, the initialization will correctly rEREAD the DATA statements.

Refer to DATA and READ for further discussion.

RESUME

At the end of the error-handling routine, place a RESUME if you want to resume the program at the command in the statement that caused the error.

Program example:

This program illustrates that RESUME restarts execution at the command in which the error occurred, here the INPUT command, the second command in statement 1000, and not with the PRINT command at the beginning of the statement. The CLRERR command is necessary to clear the error condition to prevent the ONERR routine from executing in a continuous loop, never executing the RESUME command. Since the program doesn't use the ERRNUM function to check the error number, CTL-C (which also causes the ONERR routine to execute) will not terminate the program. To END the program, enter a zero amount. To test the program first, enter a valid integer like 45 and then an invalid number like AA.

```

10 REM  resume
900 ONERR GOTO 3000
1000 PRINT "enter integer>"; : INPUT ""; amt%
1200 IF amt% = 0 THEN END
1300 total% = total%+amt%
1400 PRINT "total="; total%
1500 GOTO 900
3000 CLRERR
3050 PRINT "not integer value"
3100 RESUME

```

If you want to reexecute the entire statement 1000 after an error, change the RESUME to a GOTO 900, which resets the ONERR line number cleared by the CLRERR command and then executes statement 1000.

Helpful hints:

Refer to ONERR GOTO.

Depending on the error, you may want to repeat the command but instead GOTO to your own restart line number.

RETURN

Use the RETURN statement to transfer program execution from a subroutine back to the statement following the most recently executed GOSUB. In most cases that should be the GOSUB that invoked the subroutine.

Program example:

The RETURN program shows how to use the GOSUB and RETURN statements to implement a subroutine. The GOSUB statement transfers execution to a group of statements, beginning at line 2000, that perform the process of drawing a low-resolution line. The RETURN command transfers control back to line 1200, the statement after the GOSUB.


```

10 REM return
1000 GOSUB 2000: REM draw multi-color line
1200 PRINT "TYPE run TO DO IT AGAIN"
1990 END
2000 REM draw multi-color line
2100 GR
2200 row = 20
2300 FOR column = 0 TO 39
2400 COLOR = 16*RND(1)
2500 PLOT column, row
2600 NEXT column
2700 RETURN

```

Helpful hints:

See GOSUB, ON ... GOSUB AND POP.

RIGHT\$(string,howmany)

When you want to extract a copy of characters from the right part of a string, use the RIGHT\$ string function. The string can be a string constant, like ABC, a string variable like *name\$* or a string expression like *name\$*+ABC. The *howmany* argument, an integer value, defines how many characters to extract. If omitted, all characters to the left end of the string are selected.

Immediate-mode example:

The following shows you how to extract a copy of the five letters of the word APPLE from the string in RS.

You type RS="ADAM'S APPLE"

You type PRINT RIGHT\$(RS,5)

ADAM displays APPLE

The RIGHT\$ function extracted a copy of the right five characters, APPLE, from the string ADAM'S APPLE. The contents of *name\$* stay the same.

Program example:

The following program shows you how to right justify a series of decimal numbers so all the decimal points align.

```

10 REM right
1000 PRINT "UNALIGNED"
1100 FOR item = 1 TO 3
1200 READ number
1300 PRINT number
1400 NEXT item
2000 PRINT "ALIGNED"
2100 sp$ = " ": REM 7 spaces
2200 RESTORE
2300 FOR item = 1 TO 3
2350 READ number
2400 REM add .001 to force trailing zeroes
2410 REM the left$ drops the right digit, the 1 in .001
2500 edited$ = RIGHT$(sp$+STR$(number+1E-03), 8)

```



```
2600 PRINT LEFT$(edited$, 7)
2700 NEXT item
5000 DATA 5.55
5100 DATA 1234.30
5200 DATA 67.00
```

Helpful hints:

Refer to the other string functions STR\$, MID\$ and LEFT\$.

RND (arithmetic expression)

You use RND, the random-number function, when you need a random number. Games often use random numbers. The random-number function will return a real number in the range from zero to one. Most often it returns a decimal fraction like .4568943, so if you're storing the number, assign it to a real variable, not an integer.

Depending on the sign of the value of the arithmetic expression, BASIC will generate one of three types of random numbers.

A positive nonzero value for the arithmetic expression generates a new random number each time BASIC executes the RND statement. ADAM always starts with the same random-number series. Use a negative argument to *seed* the random number starting a different series.

For example, type the following program and notice how BASIC generates five different random numbers. You should get the same numbers, for although they are random—meaning occurring in no particular order—they are from the same predictable series of unordered numbers.

```
NEW
10 REM RANDOM NUMBER LIST
100 FOR I=1 TO 5
200 PRINT = RND(1)
300 NEXT I
RUN
```

When you want a different series of repeatable random numbers, use a negative nonzero value for the argument of the random-number function. Each negative number begins a different series, then subsequent positive arguments return a repeatable sequence of random numbers. This allows you to test your programs with the same familiar numbers. When you know the program works, delete the RND number statement with the negative argument.

Type these statements to get a ten-number list.

```
NEW
10 REM PREDICTABLE RANDOM NUMBERS
100 RN = RND(-13456)
200 FOR I=1 TO 10
300 PRINT RND (1)
400 NEXT I
RUN
```


By setting the negative seed value based on a truly random external event, like the time it takes for the operator to respond to a question, you can create a truly unpredictable random series of numbers.

When you don't want to store the number but you need the same number a second time, use a zero argument and BASIC will return the same random number the second time.

Immediate-mode example:

```
You type      PRINT RND(15)
ADAM displays  .3456789
You type      PRINT RND(0)
ADAM displays  .3456789
```

Program example:

Usually you require a random number within a range of numbers, say between 100 and 255. To convert the random number to a value within the range, multiply the random number by the range ($155 = 255 - 100$) and add the result to the low range (100). If the low range is zero you can omit the add step.

For example, suppose you want to place dots randomly on the right half of the first high-resolution screen, which has 192 horizontal rows and 256 vertical columns. The right half of the screen goes from column 128 to 255. The program will use the same random number to calculate the row and dot color.

```
10 REM rnd
1000 PRINT "press # on controller 1"
1010 IF PDL(11) = ASC("#") THEN 1050
1020 seed = seed-1
1030 GOTO 1010
1050 x = RND(seed): REM seed number
1090 HGR
1100 column = 127+128*RND(1)
1200 row = 159*RND(1): REM next number
1300 HCOLOR = 15*RND(0): REM same number as row
1400 HPLOT column, row
1500 GOTO 1100
```

ROT= (rotation value)

When you want to rotate a shape before you display it on the high-resolution screen, use the ROT statement. Values for the arithmetic expression range from 0 to 63. The following table shows the rotational effect of some values.

<i>Value</i>	<i>Rotation</i>
0	No rotation
16	DRAW will rotate the shape 90 degrees before displaying it.
32	DRAW will rotate the shape 180 degrees, turning it upside down.
48	DRAW will rotate the shape 270 degrees.

For SCALE=1 DRAW recognizes only the above four values. For SCALE=2 DRAW recognizes eight rotations.

Immediate-mode example:

See example in SCALE.

Program example:

The example shows eight rotations of the SmartBASIC default shape.

```

10 REM  rot
1000 HGR
1100 HCOLOR = 13
1200 SCALE = 4
2000 FOR degree = 0 TO 59 STEP 8
2050 ROT = degree
2100 column = 16+(degree*4)
2200 DRAW 1 AT column, 140
2300 NEXT degree
3000 FOR degree = 0 TO 59 STEP 8
3100 PRINT TAB(1+(degree/2)); degree;
3200 NEXT degree
3300 PRINT
3400 PRINT TAB(10); "ROT VALUES"

```

Helpful hints:

Unrecognized rotation values will cause DRAW to use the next smaller recognized rotation.

RUN filename,Ddrive

Use the RUN statement to tell BASIC to execute a program. The filename is optional. If omitted, BASIC executes the current program in memory beginning with the first line number. If present, BASIC will look in the tape or disk directory for a current version of the BASIC file. If it finds the file, it will LOAD it into memory and execute it. If it can't find the filename, it will display the following error message:

FILE NOT FOUND

Immediate-mode example:

To RUN the program in memory,

You type 1000 ?"HELLO"

You type . RUN

ADAM displays "HELLO"

It ran the program, which PRINTed the greeting HELLO.

To LOAD a program from type or disk cassette and execute it,

You type RUN(program name)

Program example:

You can write a menu program that calls other programs. The MAILMENU program in Chapter 10 illustrates this technique. The program examples for PEEK and POKE illustrate how one program RUNs another, as well as how to pass data from one program to another using the area protected by LOMEM:.

Helpful hints:

See SAVE, LOAD, DELETE and RECOVER.

SAVE program name,Ddrive

When you want to copy the program in memory to tape or disk, use the SAVE command. If the program already exists, ADAM moves the current entry to the backup directory.

The *program name* consists of one word of 10 characters or less. Even if you enter more than 10 characters, BASIC uses only the first 10.

Ddrive. The D identifies the parameter as a drive number. The left tape drive is 1, the right tape drive is 2, and the disk drive is 3. If the number is omitted, BASIC defaults to the drive used last.

Immediate-mode example:

Type the following single-statement program, so you have a program to SAVE. The *Companion* recommends that you identify the program name with a REM command as statement 10, so you can match a directory entry with a program listing.

```
You type      NEW
You type      10 REM save-examp
You type      SAVE save-examp,D1
ADAM displays ] when it finishes copying the program to the left tape
drive. To look at the directory,
You type      CATALOG
Adam displays VOLUME; FIRST DIR
              A 1 save-examp
```

Now SAVE the same program again.

```
You type      SAVE save-examp
You type      CATALOG
ADAM displays VOLUME: FIRST DIR
              a 1 save-examp
              A 1 save-examp
```

The lowercase letter a represents the backup version of the program *save-examp*. The current version is identified by the capital A. The one means ADAM needed only one 1024 block to store the program statements.

Helpful hints:

Remember to put the cassette on which you want to save the program into the tape drive before pressing the RETURN key.

Use the CATALOG command to make sure you have the right cassette and that there isn't already a program on the tape or disk with the same name.

If the program name already exists and you want to replace it with the updated version in memory, simply type the word SAVE and press the up-arrow key until you reach the line on which the CATALOG command displayed the name of the program. Press the space bar to remove extraneous lead characters and then use the right arrow to pass through every letter in the program name. This loads the name into the input buffer. Press the RETURN key, and BASIC will write the updated version of your program on the cassette.

It takes so long because BASIC must first read and update the directory and then write out your program. This takes a lot of time, especially when the directory and location to store your program are far apart on the tape.

It's good programming practice to save the program on two different cassettes. That way if a cassette gets damaged by heat, cold, fingerprints or a heavy foot, you have another copy. Even when you keep backup copies on another cassette, always make sure you have a printed listing of the program. The printed listing is your ultimate backup. A looseleaf binder is a good place to store your listings.

If you get an I/O ERROR, try it again. The second time ADAM will try to write your program to a different location on the tape.

SCALE= (size)

Use the SCALE command to expand the size of the shape to be drawn by DRAW or XDRAW. A value of 1 for size means no expansion of the shape, while 2 doubles the size, 3 triples it and 4 quadruples the original size. The expansion factor goes up to 255, but by then the shape disappears off the screen.

Immediate-mode example:

BASIC has a predefined shape table with one entry for a square shape. We'll use it to illustrate the SCALE command by drawing three boxes. Each subsequent box encloses the previous one.

You type	HGR
You type	HCOLOR=8 (brown)
You type	ROT=0 (don't tilt the box)
You type	SCALE=1
You type	DRAW 1 at 100,100
ADAM displays	a tiny box in the middle of the screen
You type	SCALE=2
You type	DRAW 1 at 100,100

ADAM displays a box around the tiny box
 You type SCALE=4
 You type DRAW 1 at 100,100
 ADAM displays a bigger box around the other two
 Try larger values for SCALE until the box goes off the screen.

Program example:

Here's a copy of Coleco's rotating box.

```

10 REM  scale
1000 HGR
1100 HCOLOR = 3
1200 ROT = 0
2000 FOR big = 1 TO 44
2100 SCALE = big
2200 DRAW 1 AT 128, 80
2300 HOME: PRINT "SCALE="; big
2400 FOR delay = 1 TO 400: NEXT delay
2500 XDRAW 1 AT 128, 80
2900 NEXT big

```

Helpful hints:

Refer to HGR, which displays the high-resolution graphics screen.

Refer to ROT, which rotates the shapes from 0 to 360 degrees.

Refer to HCOLOR, which defines the 16 possible colors.

Refer to DRAW, which paints the shape on the screen.

SCRN (column,row)

In low-resolution graphics mode you can find the number of the color displayed on the screen with this function. Both the *column* and *row* value can range from 0 to 39.

Immediate-mode example:

Here you PLOT a green square on the low-resolution screen. Notice that once you type GR, the text area shrinks to the bottom four lines.

You type GR
 You type COLOR=6
 You type PLOT 20,20
 You type PRINT "color=";SCRN(20,20)
 ADAM displays color=6

This is the value for green, the color you PLOTTed on the screen at column 20, row 20.

Program example:

This example uses the PDL function to move a flashing square around the screen.


```

10 REM  scrn
100 DIM shade$(15)
900 FOR hue = 0 TO 15: READ shade$(hue): NEXT hue
1000 REM  write random colors
1050 GR
1100 FOR row = 0 TO 39
1200 FOR column = 0 TO 39
1300 COLOR = 15*RND(1)
1400 PLOT column, row
1500 NEXT column
1600 NEXT row
2000 REM  read colors
2050 COLOR = 0: REM  black
2100 FOR row = 0 TO 39
2200 FOR column = 0 TO 39
2250 hue = SCRN(column, row)
2300 HOME: PRINT "color="; shade$(hue); , "VALUE="; hue
2400 PRINT "PRESS ANY KEY TO CONTINUE": GET key$
2500 IF ASC(key$) = 3 THEN END
2600 PLOT column, row
2700 NEXT column
2800 NEXT row
3000 DATA  BLACK,MAGENTA, "DARK BLUE","DARK RED","DARK GREEN"
3100 DATA  GREY,GREEN,BLUE,TAN,RED,GREY
3200 DATA  PINK,"PALE GREEN",YELLOW,CYAN,WHITE

```

Helpful hints:

Low-resolution graphics has two gray colors, numbers 5 and 10. The SCRN function always returns a 10 even though you use COLOR=5 when you display the color.

Use this function when you need to keep track of the colors of the low-resolution screen, instead of defining a separate 40 by 40 array to store the color number.

SGN (arithmetic expression)

When you want to test the sign of a number, the SGN function provides an easy way to do it.

Immediate-mode example:

The SGN function returns a minus one if the value of the arithmetic expression is negative.

You type PRINT SGN(-5*12)

ADAM displays -1

The SGN function returns a zero if the value of the arithmetic expression is zero.

You type PRINT (5-5)

ADAM displays 0

The SGN function returns a one if the value of the arithmetic expression is positive.

You type A5=23.45

You type PRINT SGN(A5)

ADAM displays 1

Helpful hints:

Suppose you need to test if all three variables A,B,C have positive amounts. You can add together the SGN function of each. If the result equals three, all were positive numbers.

SIN (arithmetic expression in radians)

Trigonometry defines the SINE of an angle as the following ratio:

$$\text{SIN } \theta \frac{\text{opposite side}}{\text{hypotenuse}}$$

The Greek letter theta (θ) represents an angle like 45 or 50 degrees. For example, consider an isosceles (equal-sided) right triangle.

*Immediate-mode example:*

One use of the formula finds the length of the hypotenuse when you know the angle and the length of the opposite side. For example, you're building some shelving in a child's room and you want to make the brackets. Type the following:

```
10 REM FIND THE SINE
1000 PRINT "Enter an Angle (expressed in radians)";
2000 INPUT R
3000 Y=SIN(R)
4000 PRINT "The Sine of A";R;"Radian Angle is";Y
5000 END
```

The SIN function requires the value of the angle to be stated in radians not degrees. In simplest mathematical terms, the radian unit of measure derives from the radius of a circle—the distance from the center to the rim of the circle. It is the angle subtended, or cut, by the arc of a circle equal in length to the radius of the circle.



Curved line has same length as radius

You can convert degrees to radians by multiplying the degrees by $\pi/180$. The Greek letter π (π) represents the ratio of the circumference of a circle to its diameter and has a constant approximate value 3.14159265.

Program example:

The trigonometry functions have use in navigation and surveying, but they also occur in nature as waves. The following program plots a sine wave curve on the video screen.

```

10 REM   sine
1000 pi = 3.14159265
1100 HGR: HCOLOR = 9
1200 HPLOT 0, 80 TO 255, 80
2000 FOR degree = 0 TO 360 STEP 2
2100 sine = SIN(degree*pi/180)+1
2300 row = 159-(80*sine)
2400 HPLOT degree/2, row
2900 NEXT degree
3000 HTAB 1: PRINT "0": HTAB 20: PRINT "360"
3100 PRINT "PLOT OF SINE - IN DEGREES"
3200 GET key$
3300 TEXT

```

Helpful hints:

See TAN.

SPC (number of spaces)

You use the SPC function only in the PRINT statement to print the number of spaces defined by the value of a numeric constant, integer or decimal variable or arithmetic expression.

Immediate-mode example:

This example places a space between city and state on a mailing-label line.

You type	city\$="ANYTOWN"
You type	state\$="XX"
You type	PRINT city\$;SPC(1);state\$
ADAM displays	ANYTOWN XX

Program example:

This example shows how to center text on the screen or on the printer. Any typist knows the formula for centering. You count the number of letters in the text you want to center. You subtract the text length from the length of the line and divide the result by two. The result is number of spaces you need

from the left-hand margin to center the text. The ADAM screen has a width of 31 characters. Use the `LEN` function to let BASIC figure the length of the text you want to center. Then plug the result of the division by two into the `SPC` function to print the necessary spaces to center the text.

```
10 REM   spc used to center a title
1000 title$ = "Tax Expenses"
1100 sleft = (31-LEN(title$))/2
1200 PRINT SPC(sleft); title$
```

Helpful hints:

Refer to the `TAB` function to print at a certain column.

SPEED=speed value

You use the `SPEED` statement to control the speed with which ADAM displays text, listing of BASIC programs and program output. When BASIC loads it sets the speed to the fast speed, 255. The slowest speed is 0.

Immediate-mode example:

You type	<code>SPEED=128</code>
You type	<code>PRINT "slow"</code>
You type	<code>SPEED=64</code>
You type	<code>PRINT "slower"</code>
You type	<code>SPEED=0</code>
You type	<code>PRINT "almost a dead stop"</code>

Program example:

This program allows you to use hand controller 1 as a paddle to control the speed of the displayed text. When you push the joystick up or away from you, the speed slows. As you pull the joystick down toward you, the speed and the beep sound increase.

The outer `FOR . . . NEXT` loop reads four message lines. The nested `FOR . . . NEXT` loop stores each character of a message line as an integer value in the array `message%`.

The print message `FOR . . . NEXT` loop sets the `SPEED` from the current value of `PDL(1)`, which returns a value between 0 and 255 depending on how long you push or pull the joystick in either a north or a south direction. Then the loop converts each integer back to a character, displays it, and uses `CHRS(7)` to make a beep sound.

When you press `CTL-C` to end the program, the `ONERR` command causes control to transfer to line 2000, which sets `SPEED=255`—the fastest speed.


```

10 REM speed
100 DIM message$(500)
500 ONERR GOTO 2000
1000 DATA "I LOVE ADAM. "
1010 DATA "I can play games, "
1020 DATA "type letters and reports "
1030 DATA "and write Basic programs. "
1300 letter% = 1
1400 FOR line = 1 TO 4
1450 READ message$
1460 FOR position = 1 TO LEN(message$)
1470 message$(letter%) = ASC(MID$(message$, position, 1))
1475 letter% = letter%+1
1480 NEXT position
1490 NEXT line
1500 REM print message
1510 FOR letter = 1 TO letter%
1520 SPEED = PDL(1)
1530 PRINT CHR$(message$(letter)); CHR$(7);
1540 NEXT letter
1900 GOTO 1500
2000 SPEED = 255

```

Helpful hints:

The *Companion* finds $SPEED=125$ a comfortable reading speed when you want the operator to carefully read the instructions the program displays on the screen.

SQR (positive arithmetic expression)

In mathematics formulas you frequently need to find the square root of a number. You use the SQR function to find what number, when multiplied by itself, equals the value in the parentheses. For example, the square root of 4 is 2 because 2 times 2 equals 4.

Immediate-mode example:

The Pythagorean theorem states that the sum of the squares of the length of the sides of a right triangle is equal to the square of the length of the hypotenuse.

$$c^2 = a^2 + b^2$$

Once you get the sum of the squares of the sides, you can use the square-root function to get the length of the hypotenuse. If one side of the right triangle has a length of 3 and the other 4, you can compute the hypotenuse as follows:

```

You type      a2 = 3 * 3
You type      b2 = 4 * 4
You type      PRINT SQR(a2+b2)
ADAM displays 5

```

The square of 3 equals 9, and the square of 4 equals 16. The sum of the squares of the sides, $9 + 16$, equals 25. The square root equals 5, the length of the hypotenuse.

Program example:

Statistical analysis often requires computing the standard deviation of a series of numbers. The standard deviation measures the average deviation from the average of all the numbers. By computing the standard deviation sample of a group, predictions about the group can be made with a certain degree of probability. That's how the television rating people figure out how many people are watching a television program. You find the standard deviation computing:

1. The average of a series of numbers.
2. The sum of the square of the deviations of each number from the average for the series.
3. The average of the sum of squares, called the variance.
4. The square root of the variance.

For example, consider three numbers 3, -4 and 7. The average is $[3 + (-4) + 7]/3$ or $6/3$ or 2. The sum of the square of the deviations from the average is shown below.

Number	-	Average	=	Deviation	Squared
3	-	2	=	1	1
-4	-	2	=	-6	36
7	-	2	=	5	25
Sum of squares of deviation					62
The variance is $62/3$					

To compute the variance and standard deviation,

You type ?62/3

ADAM displays 20.6666667 (variance)

You type ?SQR(62/3)

ADAM displays 4.54606057 (standard deviation)

The following program computes the standard deviation for any series of numbers entered.

```

10 REM  sqr standard deviation
100 DIM number(100)
900 TEXT
1000 PRINT "Enter a series of numbers"
1050 subscript% = 0: total = 0
1200 INPUT "NUMBER>"; number$
1250 IF LEN(number$) = 0 THEN 1200
1300 IF number$ = "end" THEN 2000
1320 subscript% = subscript%+1
1350 number = VAL(number$)

```



```
1400 number(subscript%) = number
1440 total = total+number
1500 GOTO 1200
2000 count = subscript%
2100 average = total/count
2130 PRINT "AVERAGE="; average
2150 total = 0
2200 FOR subscript = 1 TO count
2210 deviation = number(subscript)-average
2230 total = total+deviation*deviation
2290 NEXT subscript
2300 PRINT "VARIANCE="; total/count
2310 PRINT "STD DEV="; SQR(total/count)
```

Helpful hints:

Use the SQR function instead of raising to the power .5 because the square-root calculation is much faster.

STOP

At the place in your program you want BASIC to STOP executing instructions and let you see what is going on, put in a STOP statement. When BASIC encounters the STOP statement, it displays the] prompt and allows you to execute immediate-mode statements. To resume program execution, type a CONTinue command.

Immediate-mode example:

STOP serves no useful purpose in the immediate mode.

Program example:

The following program illustrates how to use STOP to allow you to view what the program is doing.

```
You type      NEW
You type      10 REM STOP
You type      1000 FOR ctr=1 TO 2
You type      1100 STOP
You type      1200 NEXT ctr
You type      RUN
ADAM stops at line 1100
You type      ?ctr
ADAM displays  1
You type      CONT
ADAM stops at line 1100
You type      ?ctr
ADAM displays  2
You type      CONT
ADAM displays  ]
```


At each cycle of the FOR . . . NEXT loop, the STOP statement caused BASIC to enter the immediate mode, where you used the PRINT command to view the value of the variable *ctr*. Use this technique in more complex programs to view the values used to make decisions and do calculations. It helps you uncover program mistakes. You may think a variable has a certain value when in fact you made a mistake and it does not.

Helpful hints:

The unexpected happens more often than any of us would like. Insert a STOP statement wherever you have reason to believe there might be a problem or an interruption in program logic. This prevents damaging of files or crashing the program irreparably.

STR\$(string\$)

When you need to convert a number to a string, use the STR\$ function. The string returned by STR\$ consists of the characters the PRINT statement would display. You may want to format a number differently from the way BASIC would display it. By converting the number to a string, you can use the string functions LEFT\$, RIGHT\$ and MID\$ to modify the string before PRINTing it.

Immediate-mode example:

Suppose you only wanted to print the left-hand digit of each amount.

You type money=200.244

You type ?LEFT\$(STR\$(money),1)

ADAM displays 2

STR\$(money) becomes 200.244 and LEFT\$("200.244",1) returns one character from the left side.

Program example:

This routine formats a number for dollars and cents.

```

10 REM   str format a number
100 DIM field$(14), mask$(14), work$(14)
. 200 mask$ = "000,000,000.00"
. 230 dp = 12: REM   in mask, decimal point 12 from left
1000 REM   load mask into array
1010 FOR subscript = 1 TO LEN(mask$): mask$(subscript) = ASC(MID$(mask$, subscript, 1)): NEXT subscript
1050 INPUT "amount>": amount$
1070 IF LEN(amount$) = 0 THEN END
1080 amount = VAL(amount$)
1090 GOSUB 11000
1099 GOTO 1050
11000 REM   FORMAT NUMBER
11020 IF amount > 999999999 THEN PRINT STR$(amount): RETURN
11040 field$ = STR$(amount)
11045 REM   move mask array to work
11050 FOR v = 1 TO LEN(mask$): work$(v) = mask$(v): NEXT v
11060 fd% = LEN(field$)+1
11100 FOR f = 1 TO LEN(field$)
11120 field$(f) = ASC(MID$(field$, f, 1))

```



```

11140 IF field$(f) = ASC(".") THEN fd% = f
11190 NEXT f
11200 REM    FILL DECIMALS
11210 fx% = fd%+1: REM    to left of decimal point
11220 FOR w = dp+1 TO LEN(mask$)
11230 IF fx% > LEN(field$) THEN work$(w) = ASC("0"): GOTO 11290
11240 IF work$(w) = ASC("#") AND NOT (fx% > LEN(field$)) THEN work$(w) = field
$(fx%): fx% = fx%+1
11290 NEXT w
11300 REM    FILL INTEGERS
11310 fx% = fd%-1
11320 FOR w = dp-1 TO 0 STEP -1
11340 IF fx% = 0 THEN work$(w) = ASC(" ")
11360 IF work$(w) = ASC("#") AND fx% <> 0 THEN work$(w) = field$(fx%): fx% = fx
%-1
11380 NEXT w
11400 REM    print formatted number
11420 FOR w = 1 TO LEN(mask$): PRINT CHR$(work$(w)): : NEXT w
11430 PRINT
11490 RETURN

```

Helpful hints:

Remember that the PRINT statement automatically converts a number to a string before printing. For example,

```

You type      ?STR$(money)
You type      PRINT STR$(money)
ADAM displays 200.244
You type      ?money
ADAM displays 200.244

```

Both ?STR\$(money) and ?money display the same characters, so don't use the STR\$ function alone in a PRINT statement.

TAB (column number)

You use the TAB function only in the PRINT statement to begin the next date PRINTed at the column number specified by the argument value. The column-number value can range from 1 to 255 with zero valid, but having no effect. If you use a value greater than 255, BASIC will display the error message ?ILLEGAL QUANTITY. You can only move forward to a column, not backward. For example, if you have printed 10 characters on the line, you can't use the TAB(5) to move back to column 5. You can only move forward to 11 and higher-numbered columns on the right.

Aligning data in columns makes it easier for the reader to understand. The GETKEY program example (Chapter 7) shows you how to use the TAB statement within the PRINT statement to print data in columns.

When you're designing your output, remember that the video screen TEXT line has 31 characters, while the daisy-wheel printer line has a width of 80 characters.

Immediate-mode example:

Use the TAB function to PRINT the number 30 in columns 30 and 31. The semicolon (;) between fields prevents BASIC from using its default columns to display data (see PRINT command).

You type)PRINT TAB(30);30		
ADAM displays			30
Columns	1234567890123456789012345678901		
	1	2	3

Program example:

The TAB program illustrates what happens when you tab to a column greater than the physical width of the screen or printer. BASIC just keeps printing spaces until it reaches the column specified, even though that is on another physical line. For example, on the printer, column 81 is the first column on a second physical line; on the screen, column 32 is the first column on a second line.

```

10 REM      tab
1000 SPEED  = 125
1020 PRINT
1030 PRINT TAB(0); "tab 0"
1200 FOR column = 1 TO 81
1220 PRINT TAB(column); column
1250 NEXT column
1300 SPEED  = 255

```

Helpful hints:

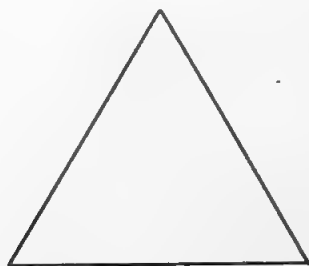
Refer to the explanation of the SPC function, which puts spaces between fields instead of beginning them at a particular column.

TAN (angle in radians)

Trigonometry defines the tangent of an angle as the following ratio:

$$\text{TAN } \theta = \frac{\text{opposite side}}{\text{adjacent side}}$$

For an angle defined in radians, the TANGent function returns the value of this ratio. The tangent approaches infinity as its argument approaches odd multiples of $\pi/2$.



Immediate-mode example:

To convert degrees to radians, multiply the degrees by the constant pi divided by 180. To calculate the TANGent of 0, 30 and 45 degrees,

```
You type      pi=3.14159265
You type      cf=pi/180 (conversion factor)
You type      ?TAN(0)
ADAM displays
You type      ?TAN(30*cf)
ADAM displays
You type      ?TAN(45*cf)
ADAM displays
```

Program example:

The TAN program uses the HPLOT command to draw a graph of the values returned by the TAN function from -2π to $+2\pi$.

```
10 REM      tan  tangent
1000 pi = 3.14159265
1100 HGR: HCOLOR = 9
1200 HPLOT 0, 79 TO 255, 79
2000 FOR degree = -360 TO 360 STEP 3
2100 tangent = TAN(degree*pi/180)
2300 row = 159-(2.67*(tangent+30))
2310 IF row > 160 THEN row = 160
2320 IF row < 0 THEN row = 0
2350 PRINT degree, tangent
2400 HPLOT (degree+360)/3, row
2900 NEXT degree
3000 HTAB 1: PRINT "-360"; : HTAB 27: PRINT "360"
3100 PRINT "PLOT OF TANGENT IN DEGREES"
3200 GET key$
3300 TEXT
```

Helpful hints:

See SIN and COS.

If you're not familiar with trigonometry but would like to learn more now that you have ADAM to do all the calculations, the *Companion* recommends you borrow a trigonometry text from the library, or buy a copy of *The Nature of Modern Mathematics*, 3d Ed., by Karl V. Smith. This college textbook is published by Brooks/Cole Publishing Company, a division of Wadsworth Inc., and may be ordered from your local bookseller or college bookstore.

TEXT

Use the TEXT command when you're in either the low-resolution graphics mode (GR) or the high-resolution mode (HGR) and you want to return to the full 24-line TEXT mode. While the SmartWriter platen provides room for

36 characters per screen line, BASIC provides room for only 31 characters on each text line.

TEXT clears the screen to black and positions the cursor to the top left of the screen, just after the] prompt.

Immediate-mode example:

After SmartBASIC loads from tape or diskette, it starts out in the TEXT mode. To change it to low-resolution graphics mode,

```
You type      GR
You type      PRINT "LINE 1"
ADAM displays  LINE 1
```

The top part of the screen goes black, and the text, LINE 1, appears on the fourth line from the bottom. To change the video display mode back to 21 lines of 31 characters,

```
You type      TEXT
You type      PRINT "LINE 1"
```

The characters LINE 1 appear on the top of the screen.

Program example:

See GR and HGR for using TEXT in a program.

Helpful hints:

If you have used a graphics mode in your program, you should place a TEXT command in your termination process, since it's likely you or someone using the program will want to be in the TEXT mode when the program ends.

TRACE

When you need to see the sequence of statement execution, use the TRACE command. It displays on the screen the line number of each statement BASIC executes. The NOTRACE function turns off the line-number display process.

Immediate-mode example:

This short program loops endlessly, but it is useful in showing you how the TRACE works.

```
You type      10 REM TRACE
You type      1000 true%=ABS(true%-1)
You type      2000 IF true THEN 4000
You type      3000 GOTO 1000
You type      4000 GOTO 1000
You type      TRACE
You type      RUN
ADAM displays #10 #1000 #2000 #4000 #4000 #10
              00 #1000 #2000 #3000 #3000 . . .
```


Press CTL-C to stop the execution. As BASIC executed each statement, it displayed the line number once when it executed it and, in the case of 1000, 3000 and 4000, again when it transferred control to it. A double line number indicates that BASIC changed the sequence of execution. By following the line numbers displayed, you can see that the ABSolute function changes the value of *true%* back and forth between one and zero, true and false. When true, BASIC executes the statement after the THEN and executes line 4000. When false, BASIC executes line 3000. To turn off the trace program execution mode.

You type NOTRACE

You type RUN

ADAM no longer displays the line numbers. Press CTL-C to stop the execution.

Program example:

You can place a TRACE within your program to display line numbers selectively from the part of the program you're having trouble with to see if BASIC actually executes them. The following program executes the TRACE and NOTRACE command to monitor whether line 5000 executes when you enter a 5.

```

10 REM trace
1000 INPUT "amount="; amount$
1100 amount = VAL(amount$)
1200 IF amount = 5 THEN 4990
1300 GOTO 1000
4990 TRACE
5000 sum = sum+amount
5005 NOTRACE
6000 GOTO 1000

```

Helpful hints:

When a program does not work as planned, you have to become a detective. You need clues or information to solve the mystery of why the program doesn't work. The first question you must ask yourself is: What information do I need to solve this mystery? Make ADAM tell you what it's doing. You can use the TRACE, STOP, PRINT, CONT and MON commands to find clues to solve the mistakes in the program. Once you have the right information, the answer to the mystery becomes obvious.

USR (value passed to machine-language routine)

You use this function when you want to execute and pass a value to a Z-80 machine-language user-supplied routine with an address in memory you have predefined. You must POKE the address of the routine into a decimal memory location (as yet undefined by Coleco).

Helpful hints:

Once Coleco makes this address available, you could use this function to call the sound routine. Instead of

POKE byte%, chip%: CALL sound%

you could write

USR(byte%)

The *Companion* expects Coleco to make this information available in a future issue of *ADAM Family Computing*.

VAL (string\$)

This function converts a character string to numeric VALues that BASIC uses in arithmetic expressions.

Immediate-mode example:

BASIC can't add the decimal number 5 to the string "12.5," but can add 5 to VAL("12.5").

You type ?5+VAL("12.5")

ADAM displays 17.5

The VAL function converted the characters "12.5" to a decimal number so that BASIC could perform the addition.

If you try to convert a letter of the alphabet to a number, the VAL function returns a zero.

You type ?VAL("ABC")

ADAM displays 0

SmartBASIC will convert as much of a string as it can.

You type ?VAL("-13.4ABC")

ADAM displays -13.4

SmartBASIC accepts a leading plus or minus sign and a decimal point to mark the beginning of the fractional part of a decimal number.

The VAL function will also convert numbers in scientific notation format.

You type ?VAL("+4.12E+2)

ADAM displays 412

The letter E indicates that the exponent to the base 10 follows. In this case, the two means ten raised to the second power, which is 100. To get the true value of a number written in scientific notation, you multiply the number by the value of the exponent. Here BASIC multiplied the number 4.12 by the value of the exponent, 100, and printed the result, 412.

Program example:

This program allows you to experiment and enter various character strings to determine how the VAL function converts each to a number.


```

10 REM  val function
1000 INPUT "AMOUNT>"; amount$
1100 amount = VAL(amount$)
1200 PRINT "VAL(AMOUNT$)="; amount
1300 GOTO 1000

```

Helpful hints:

The STR\$ function converts a number to a string.

VLIN from row, to row AT column

In the low-resolution graphics mode (GR), you use VLIN to draw a thick, colored, vertical line up and down in a particular column. The last COLOR command BASIC executed determines the color of the line. The GR command set COLOR=0 (black). A black line appears invisible against a black background. Therefore, after performing the GR command, always assign COLOR to a visible COLOR value (1 to 15) before executing the VLIN.

The low-resolution screen consists of 40 columns by 40 rows. Columns are numbered 0 to 39 from left to right. Rows are numbered 0 to 39 from top to bottom. If either the column or the row is greater than 39, SmartBASIC displays the error message:

?ILLEGAL QUANTITY

You can define the *from row*, *to row* and *column* with a number, numeric variable or arithmetic expression.

Immediate-mode example:

To illustrate the height of the low-resolution screen, the following commands draw a brown line from the top row to the bottom row at column 20.

```

You type      GR
You type      COLOR=3
You type      VLIN 0,39 AT 20

```

Program example:

The following program uses the random function to draw random-length vertical lines, at random columns and in random colors.

```

10 REM      vlin
1000 GR
2000 begrow% = RND(1)*39
2100 endrow% = RND(1)*39
2200 column% = RND(1)*39
2300 hue% = RND(1)*15
2400 COLOR = hue%
2500 VLIN begrow%, endrow% AT column%
2900 GOTO 2000

```


Helpful hints:

While SmartBASIC allows you to define the rows in reverse order—that is, *to row, from row*—you should avoid this practice because it confuses anyone, even yourself, who's trying to understand the program.

HLIN draws a horizontal line at a specific row.

PLOT draws a colored square at a particular column and row.

VTAB line

In the TEXT display mode, when you want to move the cursor to a particular line you use this command. Text lines range from 1 at the top of the screen to 24 at the bottom. In the GR and HGR modes, which allow text on the bottom four lines, the line values range from 21 to 24.

Immediate-mode example:

To position the cursor to row 24 in the current column,

You type TEXT RET

You type VTAB 20:20 RET

ADAM displays 20 (on row 20)

Program example:

The FOR . . . NEXT loop in this program uses the VTAB to PRINT two diagonal lines. The first line goes from 1 to 20 and the second from 21 to 24. Each number begins in the row that the number specifies. So you can see all 24 numbers, the GET command stops program termination until you hit any key.

```
10 REM      vtab
1000 TEXT
1100 FOR row = 1 TO 24
1300 VTAB row: PRINT row;
1400 NEXT row
1500 GET key$
```

Helpful hints:

HTAB positions the cursor to a particular column. Usually you pair HTAB and VTAB commands to position the cursor to a column, row location.

VTAB works only on the screen and doesn't affect the actual printed output. Use the PRINT command to space down lines on the printer.

WAIT port, xor-value, and value

Port defines the number of the I/O register that you want to read from or write to.

Until Coleco provides additional information, you can't use this command, which allows the program to pause until an external device returns a value indicating that it is ready to receive data. Technically, BASIC performs a logical exclusive-or with the value at the address and the xor-value. That result is added to the and-value. BASIC keeps repeating the calculations until it calculates a nonzero value, which effectively causes a wait.

Helpful hints:

The *Companion* was unable to test this command. We recommend that you don't use it.

WRITE filename,Rrecord number

With this command you tell BASIC to get ready to WRITE, or record to tape or disk. The next PRINT command lists the constants and variable actually written. You can WRITE either sequential records or, when you specify the R, parameter random records. For compatibility with Applesoft BASIC, SmartBASIC requires that you define the WRITE command as an item in a PRINT command list using value CHR\$(4) as the first item in the PRINT statement. The examples below show the correct procedure for defining and using the WRITE command.

Sequential-file example:

When you write a sequential file, the records are stored on the tape or disk, one after another in the order you write them.

The example writes a three-record file, stored as follows:

FIRST RECORD

SECOND RECORD

THIRD RECORD

After you have RUN the program, you can use SmartWriter to check the file and see that the program stored the records in this order.

To create a sequential file, you need to use four commands.

OPEN—Defines the filename and drive.

WRITE—Defines the operation as opposed to READ. It tells BASIC to direct all PRINT statement data to the tape or disk instead of the video screen.

PRINT—Writes a record consisting of the variables in the PRINT lists. It formats the variables just like it would if you were to see them on the screen, except it writes the data to the tape instead of the screen.

CLOSE—Tells BASIC you finished PRINTing to the tape and resumes displaying PRINT statement data on the screen.


```

10 REM  writeseq
500 d$ = CHR$(4)
600 PRINT d$; "mon c,o"
1000 PRINT d$; "open sequential"
1100 PRINT d$; "write sequential"
1200 FOR record = 1 TO 3
1300 READ record$
1400 PRINT record$
1500 NEXT record
1600 PRINT d$; "close sequential"
1700 PRINT d$; "nomon c,o"
2000 DATA "record one"
2100 DATA "record two"
2200 DATA "record three"

```

Random-file example:

When you WRITE to random files, you write by record number, not sequentially. The record numbers *may* occur in sequential order, but they can occur in any order—thus the term *random*. When you WRITE randomly, you must use a pair of PRINT statements. The first begins with the DS variable and defines the WRITE command and record number. The second PRINT *doesn't* begin with the DS variable and lists the data to WRITE to the file.

Random files must have fixed-length records. You specify the length using the *Llength* parameter of the OPEN statement. Even if the data you WRITE is less than the length defined in OPEN, SmartBASIC will use as much space on the tape as specified by the length parameter in the OPEN statement.

A PRINT with only a DS variable tells BASIC to output subsequent PRINT data to the screen. This allows you to display data on the screen without having to CLOSE the file, as you have to do when you write a sequential file.

This program writes the same three records, but writes them out by record number in the following sequence: 2, 1, 3. Even though they are written in random order, if you use SmartWriter to look at them you will see they are in sequential order.

```

10 REM  writerand
500 d$ = CHR$(4)
600 PRINT d$; "mon c,o"
1000 PRINT d$; "open random,L30,D1"
1200 FOR record = 1 TO 3
1300 READ record%, record$
1350 PRINT d$; "write random,R"; record%
1400 PRINT record$
1500 NEXT record
1600 PRINT d$; "close random"
1700 PRINT d$; "nomon c,o"
2000 DATA 2,"record two"
2100 DATA 1,"record one"
2200 DATA 3,"record three"

```

Helpful hints:

See Chapters 7 and 8 for sequential-file examples.

The mailing-label system in Chapter 10 READs and WRITEs records randomly to update the file.

XDRAW shape number AT column,row

When you want to erase the shape you drew with the DRAW command, you use this command. It changes every colored dot in the shape to black. For XDRAW to erase the shape correctly, it uses the same shape number and must start at the same column and row location with the same values for SCALE and ROT as when the DRAW command was executed.

Immediate-mode example:

Using the predefined square shape, with a straight line downward from the midpoint, you can DRAW and XDRAW the shape.

```
You type      HGR
You type      HCOLOR=1 (dark green)
You type      ROT=0
You type      SCALE=4
You type      DRAW 1 AT 96,100
```

ADAM displays a square with a line coming down from the midpoint at column 96, row 100. To erase the shape from the screen with black dots,

```
You type      XDRAW 1 AT 96,100
```

ADAM erases the shape from the screen. All you see is a black screen.

Program example:

The following program allows you to use hand controller 1 to move and change the size of the predefined shape.

```
10 REM xdraw
500 ONERR GOTO 9900
1000 HGR
1100 ROT = 0
1200 big = 4
1300 HCOLOR = 16
1400 GOTO 3100
3000 REM move square
3040 big = PDL(13)
3060 IF big = 15 OR big = 0 THEN big = tbig
3100 row = PDL(1)
3120 column = PDL(3)
3180 IF row > 159 THEN row = 159
3200 IF trow = row AND tcol = column AND tbig = big THEN 3000
3250 XDRAW 1 AT tcolumn, trow
3270 SCALE = big
3280 HOME: PRINT "SCALE="; big
3300 DRAW 1 AT column, row
3400 trow = row: tcolumn = column: tbig = big
3900 GOTO 3000
9900 TEXT: LIST
```

Helpful hints:

Refer to Appendix D to learn how to create your own shapes.

Refer to the high-resolution commands DRAW, HGR, HPLOT, SCALE and ROT for more examples.

APPENDIX C

List of Books and Magazines

BOOKS

This list contains the *Companion's* favorite books on personal computing, but your taste might run to other books. Be sure to visit your local bookstore, as they'll likely have an entire section on computer books. Prices change, so they aren't mentioned; if you don't find them on the shelf, most of these books may be ordered through your bookstore.

30 Computer Programs for the Homeowner, in BASIC, by David Chace. Summit, PA.: TAB Books, 1982. Many books contain BASIC program listings, but few are as practical as this one. You'll learn to write programs for a variety of directories, to collect newspaper clippings, bills and budgeting, organizers, home inventories and more.

HOW 3: A Handbook for Office Workers, 3d Ed., by Jim and Lyn Clark. Belmont, CA.: Wadsworth Publishing Company, 1982. This is a college textbook reference manual that explains how to use punctuation, capitalization, abbreviations, proper names and titles and proper grammar, and shows how to format a variety of business letters, memos, reports and other business correspondence. It also includes sections on "Words Often Misused and Confused" and "Secretarial Shortcuts." You may have to order this book at a college bookstore.

The Electronic Cottage: Everyday Living with Your Personal Computer in the 1980s, by Joseph Deken. New York: William Morrow, 1982; look for a paperback version. This is a fascinating romp through all the possibilities and potentials personal computers have in store for us. The author is intelligent and imaginative; some of his topics include "Did You Ever Play Binary Chairs?" and "Games, Hypergames and Metagames."

Microcomputer Data Communication Systems, by Frank J. Derfler. Englewood Cliffs, NJ: Prentice-Hall, 1982. This is a manual on using your computer to communicate (see Chapter 11), written clearly and simply. Learn all about modems, terminals and communications networks.

Omni Online Database Directory, by Michael Edelhart and Owen Davis. New York: Collier Books, Macmillan, 1983. Under the auspices of *Omni* magazine, this is a compendium of databases, or information resources, via telecommunications links with your ADAM (see Chapter 11).

Writing in the Computer Age: Word Processing Skills and Style for Every Writer, by Andrew Flugelman and Jeremy Joan Hewes. New York: Doubleday/Anchor Press, 1983. This book discusses the hardware and software aspects of word processing, but more important goes beyond to show how to organize and maintain electronic files, how to use communications links for researching and communicating with other authors, and various styles and techniques for writing and editing your work.

Foundations of Computer Technology, by Joseph C. Giarratano. Indianapolis, IN: Howard W. Sams Co., 1982. This large-format paperbound book explains computers in both words and pictures. It covers history, components, software and chip technology in an interesting manner.

The Computer Image, by Donald Greenberg, Aaron Marcus, Allan H. Schmidt and Vernon Gorter. Reading, MA: Addison-Wesley, 1982. This beautiful four-color book explains, in layperson's terms, all about computer graphics—but more important, it shows you how magnificent they are as an art form. This is a real coffee-table book.

Data Base Management Systems, by David Kruglinski. New York: Osborne/McGraw-Hill, 1983. While this book discusses database-management systems available for other personal computers, the principles apply to SmartFiler and the home information-management system. You'll learn how to get the most from any DBMs, whether it's Coleco's or, once CP/M is available, one of those described in this book.

The Joy of Computing, by Pete Laurie. Boston: Little, Brown, 1983. This is another beautiful, four-color coffee-table book, but packed with good information and interesting reading.

Elementary Basic: Learn to Program Your Computer in Basic with Sherlock Holmes, by Henry Ledgard and Andrew Singer. New York: Vintage Books/Random House, 1982. This is an entertaining way to learn BASIC: by solving Sherlock Holmes mysteries. A case, such as "Murder at the Metropolitan Club," is set forth, then you're given the clues in the form of a program; if you write it correctly and study the facts carefully, you solve the mystery.

Telematic Society: A Challenge for Tomorrow, by James Martin. Englewood Cliffs, NJ: Prentice-Hall, 1981. This is a revised and condensed version of Martin's 1978 book, *The Wired Society*, which was nominated for a Pulitzer Prize. Martin is one of the foremost authorities on computing and telecommunications, and he vividly describes how the two will change our lives.

Microman: Computers and the Evolution of Consciousness, by Gordon Pask and Susan Curran. New York: Macmillan, 1982. This is an idea book that shows how computers are being used in a variety of new and interesting ways, and stresses the newly forming relationship between people and computers.

The authors present the view that it's not important to distinguish a computer as a machine or a nonliving thing, since its intelligence is as important and recognizable as any living creature's.

The Naked Computer: A Layperson's Almanac of Computer Lore, Wizardry, Personalities, Memorabilia, World Records, Mind Blowers and Tomfoolery, by Jack B. Rochester and John Gantz. New York: William Morrow, 1983. This volume, coauthored by one of the *Companion's* coauthors, is a fun, funny and fascinating romp through the world of computer trivia. Comprised of thousands of short tales and anecdotes, as well as interviews, top tens in software, hardware and computer millionaires, it's for anyone who wants to learn more about computers without poring through a textbook.

The Software Catalog. New York: Elsevier Publishing Co. This massive volume, over 1,000 pages thick, contains descriptions of nearly 30,000 software packages. It's published twice a year, and you may want to reference it at your local library rather than buy it.

The Illustrated Computer Dictionary, by Donald D. Spencer. Columbus, OH: Charles F. Merrill, 1980. A good dictionary for beginners, written by a leading computer education expert. It's cross-referenced very well so you can understand the relationships between terms and functions.

The CP/M Handbook with MP/M, by Rodney Zaks. Berkeley, CA: Sybex, Inc., 1980. If you get the disk drive and become interested in CP/M, this book is an indispensable aid in understanding how CP/M works and how to use it most efficiently.

MAGAZINES

ADAM Family Computing, Scholastic, Inc., 730 Broadway, New York, NY 10003. (or through Coleco; see ADAM's literature).

Byte, Byte Publications/McGraw-Hill, 70 Main Street, Peterboro, NH 03458.

This was the first serious magazine for professionals and hobbyists. It is first-rate reading, but often quite technical; there are build-it-yourself circuits and kits and a variety of program listings.

Creative Computing, Pertec Computer Corp., 39 E. Hanover Avenue, Morris Plains, NJ 07950. *CC* is primarily devoted to hardware and software reviews, but sandwiched in between is a great deal of detailed information that helps expand your understanding of personal computers.

Enter, Children's Television Workshop, 1 Lincoln Plaza, New York, NY 10023.

This magazine is oriented toward children, computers and video games. Articles show relationships between people and computers, and even fiction.

Family Computing, Scholastic, Inc., 730 Broadway, New York, NY 10003. This was the first magazine for home computer users. It contains interesting articles and software reviews, along with program listings.

InCider, 10001001, Inc., 80 Pine Street, Peterboro, NH 03458. An Apple owner's magazine, *InCider's* main value is in its program listings, many of which will run on ADAM.

Infoworld, Popular Computing, Inc., 1060 Marsh Road, Menlo Park, CA 94025. This is the *Time* magazine of personal computing, filled with news about the people, the products and trends in the field. It also has the best hardware and software reviews to be found. Regular columnists share their views on a variety of topics.

Nibble, Micro-Spare, Inc., P.O. Box 325, Lincoln, MA 01773. Again, an Apple user's magazine, but it's devoted to program listings you may be able to use.

Popular Computing, McGraw-Hill, Inc., 70 Main Street, Peterboro, NH 03458.

How to Create a Shape Table

SHAPES

Look closely at your video screen. Notice that the apparently solid figures you view from several feet away consist of smeared dots, or very short lines of white or colored light, when viewed from a few inches away. These dots are called *pixels* and form a shape or figure on your computer screen in the same way they form pictures on your television.

SmartBASIC lets you define shapes in a shape table and then use the DRAW command to select the shape by number and have BASIC draw it on the screen starting at a column and row coordinate. For example, SmartBASIC has a built-in shape table that defines one shape, a square with a line extending down from the midpoint of the square to the bottom edge. You can DRAW the shape on the screen at any *row, column* location. For example, to draw a shape at column 40, row 107,

You type HGR (high-resolution display mode)
You type HCOLOR=7 (define color white)
You type ROT=0 (rotation)
You type SCALE=1 (size)
You type DRAW 1 AT 40,107

To erase the shape,

You type XDRAW 1 AT 40,107

The HCOLOR command defines the color. ROT lets you specify a rotational factor. In other words, you can tell BASIC a number and it will draw the shape upside down, lay it on its left side, tilt it at a 45-degree angle, etc. The SCALE command lets you tell BASIC, for example, to double or triple the size of the shape it draws on the screen. While you can use SmartBASIC's built-in shape to learn how the high-resolution command works, you'll want to learn how to design and define your own shapes for BASIC.

ADAM'S COMPANION
DEFINING A SHAPE

BASIC draws a shape on the screen in much the same way that you might draw a diagram on a piece of graph paper. Say a friend called you on the phone to give you directions to her house. "Go north on Western Avenue two miles until you see the McDonald's, then turn left," your friend says. You would draw a vertical line on the paper representing Western Avenue, and then you draw a horizontal line representing the left turn at McDonald's. Drawing video shapes works the same way; you give SmartBASIC instructions, in the form of numbers, that tell it to draw up, left, down and right. When the SCALE is 1 SmartBASIC draws a dot; otherwise it draws a line of dots.

To illustrate how to define a shape, the *Companion* will show you the instructions needed to have SmartBASIC draw the letter T. First, you design your shape in the form of dots. The five dots below form the letter T.

ooo
o
o

Imagine SmartBASIC has a crayon and your instructions tell it how to connect the dots. You must decide from which dot you want to begin drawing the shape. While you could start with any dot, choose a dot that minimizes the number of instructions needed to draw the shape. The following instructions start with the top left dot and define how to connect the dots. *Plot* means draw a dot or line connecting the current location to the next location. *Don't plot* means move from one dot to the next without drawing a connecting dot or line. The first instruction connects two dots causing two dots to appear. The second instruction extends the top of the T with another dot. The third instruction doesn't plot, but just moves the crayon back to the middle dot so the fourth and fifth instructions can draw T's stem.

<i>Instruction</i>	<i>Example</i>
1. Plot a dot, go right	oo
2. Plot a dot, go right	ooo
3. Don't plot, go left	move back to middle
4. Plot a dot, go down	ooo o
5. Plot a dot, go down	ooo o o

ADAM only understands instructions as groups of ones and zeros. Since there are only four directions (up, down, right and left) and two plotting

conditions (*plot* or *don't plot*) you can use a group of three bits to represent the eight possible combinations.

<i>Decimal code</i>	<i>Binary code</i>	<i>Action</i>
0	000	don't plot, go up
1	001	don't plot, go right
2	010	don't plot, go down
3	011	don't plot, go left
4	100	plot a dot, go up
5	101	plot a dot, go right
6	110	plot a dot, go down
7	111	plot a dot, go left

The right-hand bit represents the plot instructions (1 = plot, 0 = don't plot). The two left-hand bits represent the direction.

The following table shows the binary instructions needed to form the letter T.

<i>Decimal code</i>	<i>Binary code</i>	<i>Action</i>
5	101	plot a dot, go right
5	101	plot a dot, go right
3	110	don't plot, go left
6	010	plot a dot, go down
6	011	plot a dot, go down

After designing your shape on graph paper, you must write a list of these triplets to tell the DRAW command how to draw your shape on the high-resolution screen.

The SHAPE program in Figure D-1 (next page) uses these values in the DATA statement to draw the letter T. It changes the shape-table address, so you can no longer use ADAM's square shape unless you reload SmartBASIC or POKE it back. To display the two bytes of the address of ADAM's default shape,

You type ?PEEK(16767),PEEK(16766)

ADAM displays two decimal numbers. You may want to copy these numbers down, if you will want to POKE them back to use ADAM's default shape after RUNning the SHAPE program.

To display the decimal address defined by these two numbers,

You type adam=PEEK(16767)*256+PEEK(16766)

You type ?adam

ADAM displays the address of the default shape table.

To print ADAM's shape table,

You type FOR a=adam TO adam+13:??peek(a);spc(1);:next a

ADAM displays 1 0 4 0 54 63 36 36 45 45 54 54 63 0

In the next few pages you'll learn what these numbers mean. Now enter the SHAPE program and RUN it to display the stick figure letter 'T'.

```

10 REM      shape - leave 1000 bytes for shape table
20 LOMEM :29000
100 POKE 16766, 96
110 POKE 16767, 109
200 size = 7: REM      size of shape table
1000 REM      load shape table
1010 FOR address = 28000 TO 28000+size-1
1020 READ byte%
1030 POKE address, byte%
1040 NEXT address
1100 HGR
1110 HCOLOR = 7
1120 SCALE = 6
1130 ROT = 0
1400 DRAW 1 AT 120, 90
1900 END
2000 DATA 01,00,04,00,237,54,0
    
```

FIGURE D-1 SHAPE Program.

GROUPING THE PLOT CODES INTO BYTES

Computer memory has bits in groups of eight called bytes. To save space and quickly display the shape, you need to fit the three-bit groups of codes into bytes. Three into eight doesn't go evenly, so only two will fit and maybe a third. The diagram below divides an eight-bit byte into three sections, showing how to store the codes.

Section	C		B			A		
Bit	7	6	5	4	3	2	1	0
	M	M	P	M	M	P	M	M

M = Movement bit

P = Plot/don't-plot bit

A third code fits by assuming it is a *don't plot* and adopting the following rules:

1. You place only nonzero *don't-plot* direction codes in section C.
2. Zero in section C means ignore it—don't plot and don't move.

Plotting the shape ends when BASIC reaches a byte with a zero value. This means you can't have three *don't plot, go ups* (code=000) in the same byte because BASIC will think it has reached the end of the shape. One bit in the byte must be set to 1 to make BASIC continue plotting.

The codes for letter T fit into three bytes from right to left as follows:

Section	<u>C</u>		<u>B</u>			<u>A</u>			<u>Decimal Value</u>
Bit	7	6	5	4	3	2	1	0	
	M	M	P	M	M	P	M	M	
Byte 1	1	0	1	0	1	1	0	1	237
Byte 2	0	0	1	1	0	1	1	0	54
Byte 3	0	0	0	0	0	0	0	0	0

The third code was a nonzero *don't plot*, so it fits into section C of byte 1. The zero-value last byte marks the end of the shape.

DESIGN THE SHAPE TABLE

You store the bytes that define a shape in the memory shape table. The shape table consists of three parts:

1. Shape count, which tells BASIC how many shapes in the table.
2. Shape index, a relative pointer to where each shape resides—its home—in the shape table.
3. Shape bytes, which store the plotting and direction instructions.

The shape-count byte defines how many shapes are in the table. If you have only one shape in the table, its value is one. You have a maximum of 255 shapes in the shape table. The byte after the count has a value of zero and is not used, but must be present.

There is an entry in the shape index for every shape. The index points to the relative address from the beginning of the shape-definition bytes. It consists of two bytes—a low-order byte and a high-order byte. For relative addresses less than 256, the high-order byte has a value of zero and the low-order byte has the value of relative offset. For relative addresses 256 and greater, do the following:

1. Compute the decimal value of the high-order byte by dividing the relative offset by 256. Drop the decimals and take the integer or whole number as the high-order byte.
2. Compute the low-order byte by multiplying the high-order byte by 256 and subtracting the result from the relative offset. This will be a number less than 256.

For example, suppose the shape begins 1000 bytes from the beginning of the shape table.

1. Dividing 1000 by 256 gives 3.90625. Dropping the decimals gives 3 as the high-order byte.

2. Multiplying 3 by 256 gives 768. Subtracting this from 1000 gives 232, the decimal value of the low-order byte.

With only the letter T shape, the table would have the following values:

<i>Relative Byte</i>	<i>Decimal Value</i>	<i>Description</i>
0	01	shape count
1	00	unused
2	04	index low-order byte
3	00	index high-order byte
4	237	first shape byte
5	54	
6	00	end of shape

Adding another shape would add another index entry and shift the first shape byte to relative byte 6.

With two shapes the table looks like this:

<i>Relative Byte</i>	<i>Decimal Value</i>	<i>Description</i>
0	02	two shapes in table
1	00	unused
2	06	low-order byte of index to first shape
3	00	high-order byte of index to first shape
4	09	low-order byte of index to second shape
5	00	high-order byte of index to second shape
6	237	beginning of first shape byte
7	54	
8	00	end of first shape
9	25	beginning of second shape
10	37	
11	00	End of second shape

You can enter the following statements to change the SHAPE program to use this shape table to draw a T and a square.

200 SIZE=12: REM size of shape table

1500 DRAW 2 at 150,90

2000 DATA 01,00,06,00,09,00,237,54,00,37,55,00

STORING THE SHAPE TABLE IN MEMORY

You store a shape table in memory after the end of the BASIC interpreter program and before the symbol table. Use the LOMEM: command as the first

statement in your program to make room for the shape table. The *Companion* recommends that you assume the end of BASIC at the decimal memory location 27999 and load your shape table beginning at decimal memory location 28000. So that BASIC's symbol table doesn't destroy your shape table and vice versa, set LOMEM: to a decimal location 28000 plus the size of your shape table. The examples in the *Companion* set low memory to 29000, providing 1000 characters for the shape table. The following statement sets LOMEM:. Place the LOMEM: command as the second statement in your program after the REM that identifies the program.

```
10 REM program name
```

```
20 LOMEM:29000
```

BASIC needs to know where you stored your shape table. You must POKE the decimal values of the low and high hexadecimal address of the first byte of the shape table into decimal locations 16766 and 16767. In other words, follow these recommendations.

1. Divide the location of the shape table by 256.

```
You type      ?28000/256
```

```
ADAM displays 109.375
```

2. Disregard the decimals and take the integer value or whole number (109) and POKE it into 16767.

```
20 POKE 16767,109
```

3. Multiply the integer value from step 2 by 256 and subtract it from the location of the shape table.

```
You type      ?28000-(109*256)
```

```
ADAM displays 96
```

4. POKE the decimal value of the low-order byte of the hexadecimal into 16766.

```
30 POKE 16766,96
```

If you always use 28000 as the location of your shape table, you can use the statements in the SHAPE program to load and define the location of your shape table.

WRITE A SHAPE-MAKER PROGRAM

Now that you have gone through the process of defining a shape, you realize that you need a shape-maker program to help you define your shapes. The SHAPEMAKER program shown in Figure D-2 (next page) relieves you of the tedious calculations required to determine the numerical instructions that define the shape. You use keyboard arrow keys to define the direction and function keys I and II to specify just movement or plotting and movement. The PRINT key will print out the shape-table values. The CLEAR key erases the shape on the screen and the shape in memory. CTL-C terminates the program and prints the shape values.


```

10 REM      shapemaker
20 LOMEM :29000
200 DIM byte%(200)
300 GOSUB 8500: REM  initialization
1000 REM    main process
1050 HOME: PRINT "USE ARROW KEYS"
1100 GET key$: key% = ASC(key$)
1110 IF key% = 3 THEN GOSUB 3000: END
1120 IF key% = 149 THEN GOSUB 3000: GOSUB 1050
1130 IF key% = 150 THEN GOSUB 8500: GOTO 1050
1150 IF key% >= 160 AND key% <= 163 THEN 1200
1160 PRINT beep$: GOTO 1050
1200 direction% = key%-160
1300 HOME: VTAB 23: PRINT "  I  "; "  II "
1310 PRINT "MOVE PLOT"
1320 GET key$: key% = ASC(key$)
1330 IF key% = 129 OR key% = 130 THEN 1400
1340 PRINT beep$: GOTO 1300
1400 dot% = (key%-129)*4
1500 byte%(subscript%) = dot%+direction%
1520 max% = subscript%: subscript% = 1
1600 GOSUB 2000: REM  load shape table
1610 GOSUB 2100: REM  draw shape
1620 subscript% = max%+1
1990 GOTO 1000
2000 REM      load shape table
2010 address = 28004: subscript% = 1: size% = 3
2020 a% = byte%(subscript%): subscript% = subscript%+1
2030 b% = byte%(subscript%)*8: subscript% = subscript%+1
2040 c% = byte%(subscript%)
2050 IF c% > 3 THEN c% = 0: GOTO 2070
2060 c% = c%*64: REM  shift left 6 bits
2065 subscript% = subscript%+1
2070 byte% = c%+b%+a%
2075 POKE address, byte%: address = address+1
2080 IF subscript% < max%+2 THEN 2020
2090 size% = address-28000: RETURN
2100 REM      draw shape
2110 HGR: ROT = 0: HCOLOR = 7
2120 SCALE = big%: DRAW 1 AT 120, 80
2500 HOME: PRINT "CHANGE SCALE? USE KEY PAD 1"
2510 tbig% = PDL(13): IF tbig% = 15 THEN 2510
2530 IF tbig% THEN big% = tbig%: GOTO 2100
2900 RETURN
3000 REM      print
3010 PR #1: GOSUB 2000: REM  load shape table
3015 PRINT "REL VALUE"
3020 FOR location = 28000 TO 28000+size%
3030 PRINT location-28000; TAB(5); PEEK(location)
3040 NEXT location
3080 PRINT CHR$(4): REM  turn off printer
3090 RETURN
8500 FOR subscript = 1 TO 200
8510 byte%(subscript) = 0
8520 NEXT subscript

```



```

8530 HGR: subscript% = 1
8540 true% = 1: beep$ = CHR$(7)+CHR$(7): big% = 4
8570 POKE 16766, 96: POKE 16767, 109
8600 RESTORE: FOR address = 28000 TO 28200
8620 IF address > 28002 THEN POKE address, 0: GOTO 8640
8630 READ byte%: POKE address, byte%
8640 NEXT address
8650 DATA 01,00,04
8690 RETURN

```

FIGURE D-2 SHAPEMAKER Program.

The PRINT, CLEAR and CTL-C work only when the program displays the message

USE ARROW KEYS

When the program displays

CHANGE SCALE? USE KEY PAD 1

use controller 1 keypad to change the size of the shape. Press the 0 pad if you don't want to change the size.

WARNINGS: You must enter *two* plot instructions to see anything; the first PLOT doesn't appear on the screen until you enter the second. If you enter, one after the other, two or more up arrows and MOVEs, you may cause the creation of a zero byte ending the shape. Use the CLEAR key and start again, this time using a different sequence of instructions that don't require two or more up arrows and MOVEs in sequence.

MANAGING A SHAPE TABLE

In addition to defining a shape with numbers, a shape-maker program should allow you to manage or manipulate a shape table. Managing a shape table consists of loading the shape table using BLOAD, displaying the shape, adding shapes to the table, replacing changed shapes, deleting shapes, and saving the shape table using BSAVE. To add these features requires quite a lot of programming. The *Companion* has given you the skills to tackle this project on your own.

Although similar to the picture-maker program, the shape maker should allow you to create and change *shapes*, such as the letter T and the square, instead of creating low-resolution pictures. Shapes are useful for displaying figures like a solid-colored circle, which is impossible to draw with low-resolution squares.

How to Start Your Own Computer Club

Taylor Barcroft and his son, Adam, formed the first ADAM computer club, called the First Southern California Adam Users Group. They are also the first to publish an ADAM newsletter, called *The Garden of Adam Newsletter*. Taylor created so much interest with his computer club and newsletter that he was quoted in *USA Today*. Taylor and Adam hope ADAM computer clubs, or user groups, will spring up all around the country, so they shared their experiences with us to help you start one in your community.

People join a computer club because they want to learn more about their computer, to swap program listings and software, and to tell others about their own experiences—good and bad. Usually, one or two people start the club; Jonathan Rotenberg held the first meeting of the Boston Computer Society in his bedroom when he was 14 years old, with one friend in attendance. Today, six years later, the BCS boasts over 20,000 members, has over 30 user groups and special-interest groups, and publishes a bimonthly magazine, *Computer Update*.

The first thing to do is find out who, besides yourself, has an ADAM. You can ask the store where you bought yours, put notices on the bulletin board at school or place a free or inexpensive ad in the local weekly shopper newspapers you see in grocery stores. Make a list of owners—on your ADAM, of course; use SmartWriter (or SmartFiler if you have it). At the same time, begin a list of media people who might be interested in your activities. Include the editor of your school newspaper, the local newspaper's computer writer or columnist, local radio talk-show hosts and television people such as the consumer news reporter and the "evening magazine" hosts. Include magazines such as *ADAM Family Computing* and others listed in Appendix C. The local Coleco sales representative and service technician may be interested in attending. Add anyone you can possibly think of who is interested in computers. Once your club begins holding meetings, ask people over and over who they might add to this list. You want to get the word out that you have an ADAM user group so others can join.

If at first you only find one or two ADAM owners, give them a call and ask if they'd like to get together. Once you have four or five members, you'll probably want to write meeting notices on your ADAM and mail them out.

You probably won't conduct much business at your first meeting, but try at least to select a name. One that refers to your town, geographic area or a recognizable name or landmark, such as "The Lake Forest ADAM Users Group" or "Grand Canyon Area ADAM Computer Club" or "ADAM Users of Gainesville," helps others know where you are (especially if the national magazines publish a listing).

By the third or fourth meeting, when you have enough members, it's a good idea to select officers, decide whether you're going to have dues and membership fees and make some assignments. In addition to a president, vice-president, secretary and treasurer, you may want to have some of the following positions:

- Vice-presidents for hardware and software, responsible for keeping the membership informed of all new products in each area; in addition, the software v-p should maintain a library of all the software available, including programs and listings written by members.

- A vice-president for telecommunications, to help members with modems, information utilities, databases and computerized bulletin board (CBB) services (you may, in fact, want to start one of your own), and anything else pertaining to ADAMNET.

- Newsletter editor and publisher.

- Membership coordinator, responsible for maintaining membership as well as locating, soliciting and contacting new members.

- Program coordinator, responsible for locating speakers and setting up special events for the club.

- A Coleco liaison, a single person all members route complaints and suggestions through (Coleco wants your feedback and will appreciate your selecting one person to speak for your group; ask your local Coleco rep who to contact at Coleco headquarters).

It's important to have a main event at each meeting, followed by group discussions of whatever sort people want. But remember, there should be a central purpose to motivate attendance. Invite guest speakers, perhaps a local teacher or educator who uses an ADAM. Have a Buck Rogers competition with awards for the best score. Invite game developers, communications experts, programming pros, a video consultant to demonstrate and discuss monitors. Invite the media people not only to cover your activities but as speakers, too. Be creative; programs are the most fun, and probably the most important aspect, of your club.

Once the main event is over, you may want to have a snack break, then disburse into groups such as programming, SmartWriter, SmartFiler, communications or whatever else the members want. Some club members bring

their computers to meetings to try new techniques and debug programs; others prefer just to talk. You should probably let the membership cast a vote to decide what they want.

If you're meeting at someone's home, you'll probably quickly outgrow the space. The Homebrew Computer Club was the first such group, and they met in Steve Wozniak's garage; that's where the Apple computer was born. You may want to have your meetings in a school cafeteria, auditorium, lodge meeting hall, savings and loan association meeting room or something similar. Discuss this with your members; most likely someone knows of a place, or knows someone who could help find a place, where you can hold your meetings for little or no charge. Once you've found a home, try to have your meetings on the same day each week, say every Wednesday at 7:00; people will plan their schedules around a fixed meeting if they know when it is.

If you have more questions about starting an ADAM user group or newsletter, Taylor would love to hear from you. You may write or phone:

Taylor Barcroft
First Southern California Adam Users Group
P.O. Box 599
Venice, CA 90294
(818) 355-9632

You can subscribe to Taylor's *Garden of Adam Newsletter*, too. It's \$12 for 12 issues, or \$1.50 for a sample issue.

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES



THE INDISPENSABLE COMPANION FOR THE ADAM™ FAMILY COMPUTER SYSTEM

Written specifically for use with the ADAM™, ADAM'S COMPANION takes nothing for granted, anticipating every question—from how to set up the new computer to designing original programs. It shows with actual examples, how to:

- Avoid the "common bloopers" that can confound and frustrate the beginner.
- Write a useful and practical program—and learn BASIC language in the process.
 - Create your own graphics, design your own video game, or compose and play music on the ADAM™—even if you've never used a computer before.
- Turn your computer into a word processor and type letters, term papers, resumes, etc. You can make all the corrections, add or delete words and rearrange paragraphs on the screen, and then print out a perfect copy—or several variations of the same copy—in minutes.
- Use a program that will inventory your household for insurance purposes, compile recipes, create a mailing list, or catalog your book, record, and stamp collections.
- Become familiar with other popular programs, including a dictionary, financial and household management packages, and all the latest games.

Also included:

- A chapter giving you the latest ADAM™ software and hardware on the market as well as an advance look at what is about to be released.
- A Glossary of Computer Terms • BASIC Language Definitions

... AND MUCH MORE

**ATTENTION COLECOVISION™ OWNERS...FIND OUT ALL YOU CAN DO IF YOU CONVERT
YOUR GAMEPLAYER INTO A FULL COMPUTER WITH THE ADAM™ MODULE.**



ISBN 0-380-87650-7

